Personal Computer

FUJITSU

FM77

相場多聞＝著

ナツメ社

ディスクファイル入門[1]



Personal Computer FUJITSU

FM77

相場多聞＝著

ナツメ社

ディスクファイル入門[1]



まえがき

本書は、FM—77に内蔵されているディスクの基本操作とその活用法について解説したものです。

ビジネスショー、マイコンショーなどでのデモンストレーションは、多くのグラフィックデータをこのディスクに保存し、それをいろいろな料理法によってグラフィック図形をディスプレイに表示させ、私たちの目を楽しませてくれるのです。

デモンストレーションを見ていると私たちを夢の世界に引き入れてくれると同時に、ともすると、パソコンさえあればどんなことでもできそうな錯覚におちいらせます。

ディスク内蔵タイプのパソコンを使えば、確かに相当なことができます。しかし、そこまでいくにはある程度の努力が必要です。別におどかすわけではありませんが、事実です。

また、情報化時代にパソコンを活用しようと思うと、どうしても多量のデータを扱うことになりますから、ディスク操作に関する知識は不可欠です。一方最近、ディスク内蔵タイプのパソコンが増加する傾向にあり、時代の要求を端的に物語っています。

現在のみなさんは、どんなことをやってみたいとお考えですか？

集計業務、在庫管理、顧客管理、グラフィックスなどなど、その目的はいろいろあると思います。しかし目的は違っても、考え方の基本はみな共通しています。

そこで本書では、こういった時代背景をふまえて、具体例にそくしてデータの扱い方をやさしく解説し、パソコン実用化を目ざす方々のための道しるべとなればと考えています。

本書によってぜひディスク操作の基本をマスターし、実務面で役立てていただけることを期待します。

なお本書では、F—BASICの基本的なコマンド、ステートメント等についてはあまり説明を加えておりませんので、BASICプログラミングをまだ勉強されてない方は、次の本を終えてから本書に進まれることをおすすめします。

FM-77　プログラミング入門（ナツメ社）

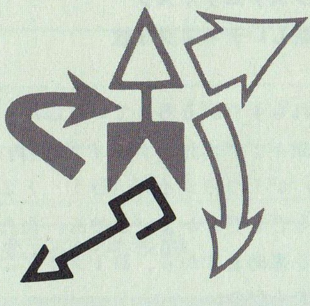
最後に、原稿整理でお世話になったパソコン編集部の方々に御礼申し上げます。

相場多聞

もくじ

第1章

ディスクの基本操作



1・1　準備するもの

あなたはいま、FM－77を前にしてどんな気持でしょうか？

これから徐々にこの本を読み進むにつれて、あなたのディスクへの理解が深まり、仕事に、そして趣味におおいに生かすことができると思います。

まず、必要な装置の確認です。

❶　FM－77本体
❷　ディスプレイ
❸　システムディスク
❹　新しいディスク3枚

〈注〉プリンタへの出力も多少扱いますので、あれば便利ですが、なくてもかまいません。お持ちでない方は画面表示へ切り換えてください。

これら4つがそろっていればだいじょうぶです。

FM－77本体はドライブ(四角い3.5インチディスクを差し込むところ）が1つのタイプ（D1）と2つのタイプ（D2）がありますが、どちらでもでもかまいません。ただし本書は、D2タイプにそくして説明を進めますから、D1タイプご使用の方は注意書きに従って読み進めてください。

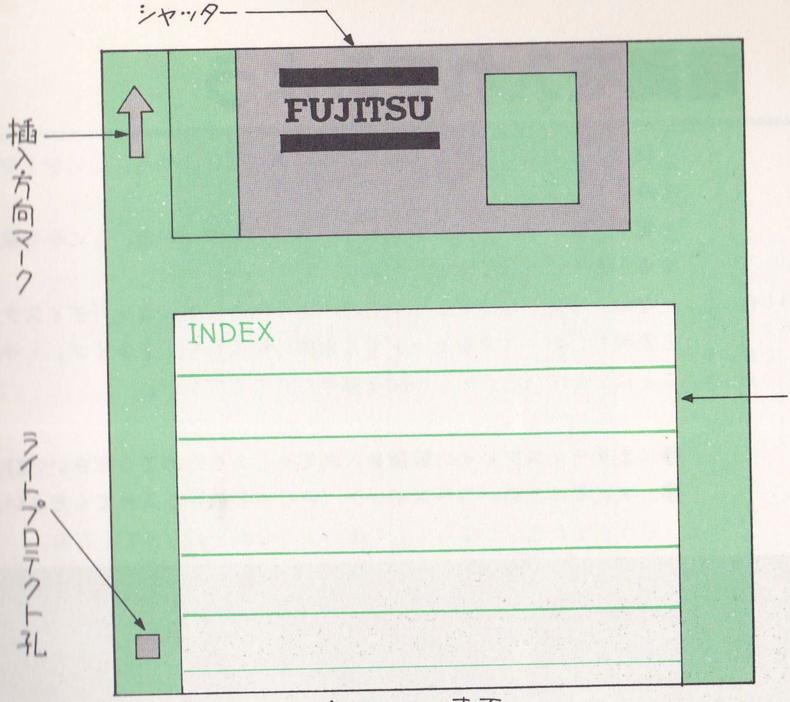
ディスプレイはカラーでもモノクロでもどちらでもかまいません。

システムディスクとは、FM－77本体購入時に標準添付されている四角いディスクのうち「F－BASIC」と書かれたラベルが貼ってあるディスクです。この中には、ディスク操作を可能にするための機械語プログラムとか、そのほか便利で実用的なプログラムが記録されていますから大切にしてください（イメージ的にはレコード盤にいろいろな曲が記録されているのを思い浮かべてください)。

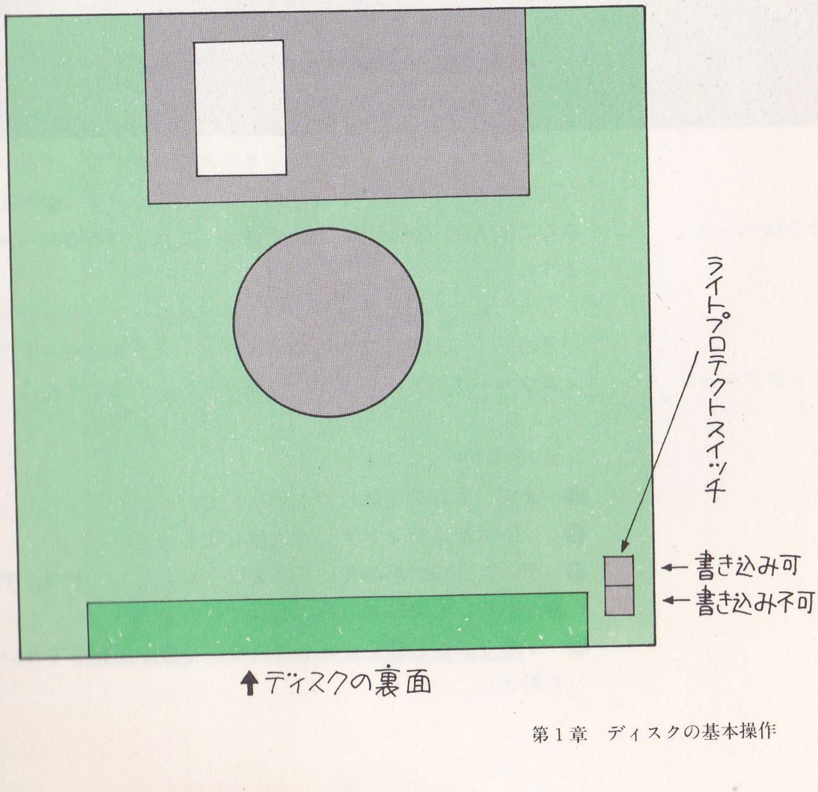
新しいディスク（3.5インチのマイクロフロッピィディスク）はどこのメーカーのものでもよいのですが、できたら富士通の標準のものがよいでしょう。

マイクロフロッピィディスクの各部名称を右ページに示しますので参考にしてください。

なお本章の記事は、『FM－77　プログラミング入門』と多少重複するところがありますが、これは本書から勉強される読者もいると考えられるためですのでご了承ください。



↑ディスクの表面



↑ディスクの裏面

1·2 電源を入れてみよう

何はともあれ、電源を入れなければ事は運びません。とにかくやってみましょう。

電気に弱いという方、コンピュータアレルギーの方、とにかく説明するとおりにやってみてください。

ディスク
ドライブ

〈注〉❷の操作に移る前に、本体前面にあるボタンのうち、「ＢＡＳＩＣ」と書いてあるボタンを押しておいてください。

なお、本書では、3.5インチの小さな四角い磁気盤を**「ディスク」**と表現し、ディスクをセットする本体の挿入口を**「ドライブ」**と呼ぶことにします（右ページの図を参考にしてください）。

❶ **まずディスプレイの電源を入れて少しあたためてください〈注〉。**
❷ **次に本体左のパワースイッチ（ＰＯＷＥＲ）を入れてください。**

どうですか？　しばらくして次のような画面になりましたね。

```
FUJITSU F-BASIC Version 3.0
Copyright (C) 1981 by FUJITSU/MICROSOFT
30530 Bytes Free

Ready
■
```

これはディスクを使わないときの初期画面です。ＦＭ－７、ＦＭ－ＮＥＷ７などを、オプションのディスクをつけずに使用したときは、みなこれと同じ画面だったはずです（これを**「ＲＯＭベース」**といいます）。

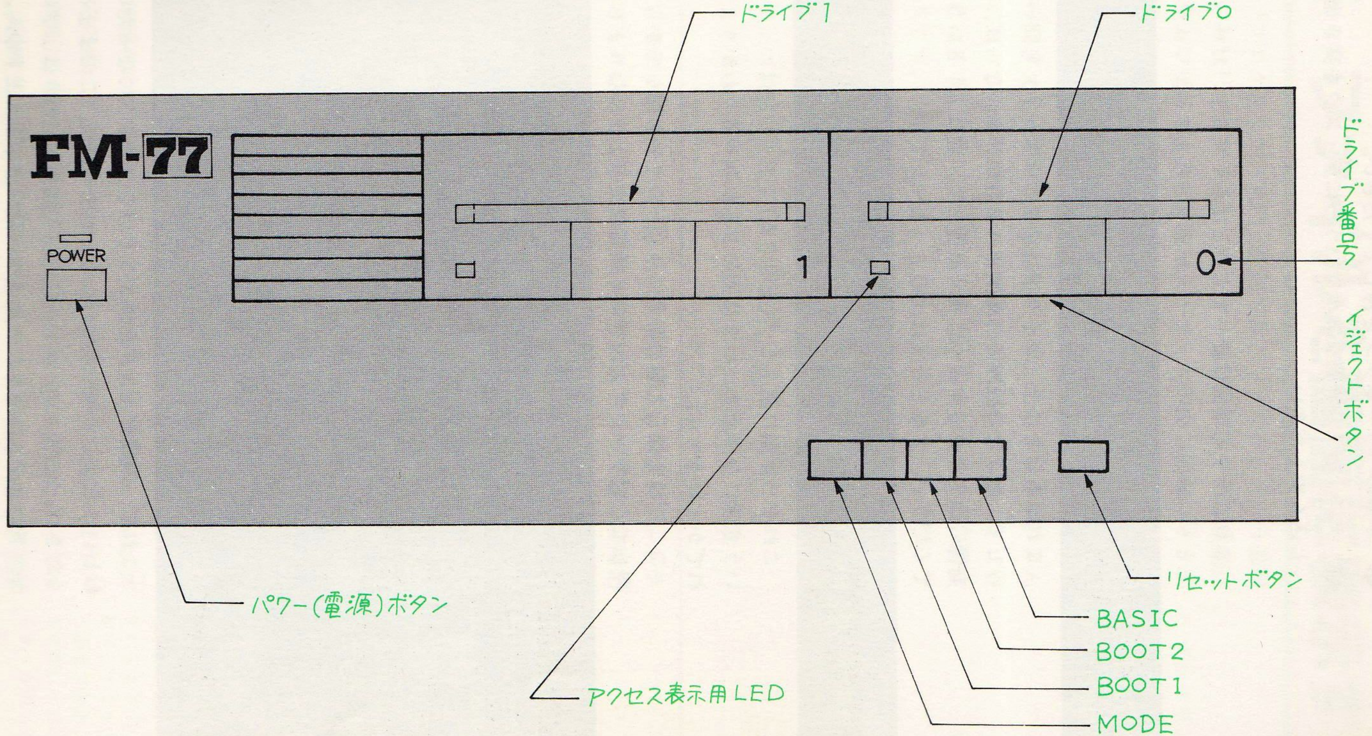
ＲＯＭベース

しかし私たちは、せっかくディスクユニット（ディスク装置）内蔵のＦＭ－７７を使っているのです。ディスク使用モード（これを**「ディスクベース」**といいます）で操作したいものですね。

ディスクベース

それではやってみましょう。

❶ **まず、本体の電源だけを切ってください。**
❷ **「システムディスク」を用意してください。**
❸ **ディスクの文字の書いてあるほうを上にし、かつ、ディスク上に書かれている矢印の方向に向けてください。**
❹ **ドライブ０と書いてある挿入口にそのまま半分くらい差し込んでください。**


ドライブ1
ドライブ0
ドライブ番号
イジェクトボタン
FM-77
POWER
1
0
パワー(電源)ボタン
リセットボタン
BASIC
BOOT2
BOOT1
MODE
アクセス表示用LED

❺　本体のパワーボタンを押してください。

❻　次に、あまり時間をおかないうちにディスクを突き当たるまで押し込んでください。

これで前半の準備は終りです。しばらくすると「ピーッ」という音とともに画面に次のメッセージが表示されますね。

```
DISK VERSION
How many disk drives      ? ■
```

これはあなたに「ドライブ数はいくつですか？」と問いかけているのです。あなたの機種の内蔵ディスク数が「2」ならば、キーボード右側の数字キーの中の 2 キーを押して、次に ⏎ というキーを押してください。すると今度は次のメッセージです。

```
How many disk files(0-15)? ■
```

これを日本語に訳すと「(使用する)ファイル数はいくつですか」という意味になります。いまは何のことかさっぱりわかりませんが、それでかまいません。

ここでは先ほどと同じように、 2 のキーを押してから ⏎ キーを押してください。ディスプレイ画面は次のようになりました。

```
DISK VERSION
How many disk drives      ? 2
How many disk files(0-15)? 2

FUJITSU F-BASIC Version 3.0
Copyright (C) 1981 by FUJITSU/MICROSOFT
25504 Bytes Free

Ready
■
```

これで、内蔵ディスクを使うことができるようになったのです。くわしいことはよくわからないかもしれませんが、このまま次へお進みください。また、コンピュータ操作を終りたいときは、ディスク挿入口の下のボタンを押すとディスクが飛び出してきますから、これを抜きとってから、本体、ディスプレイの電源をＯＦＦにしてください。

1・3　ファイルを画面に表示する

私たちは「プログラミング入門」段階では「ファイル」という表現を使わずに、単に「プログラム」と呼んでいましたが、これは正確には**「プログラムファイル」**と呼びます。

プログラムファイル

ファイルにはこのほか**「データファイル」**というのがありますが、これは徐々に説明します。

私たちがいまディスクを立ち上げるのに使ったシステムディスクには、いろいろなファイルがおさめられています。ここではまず、どんなファイルが入っているのか、これを画面に表示させてみることにします。それには**FILES（ファイルズ）コマンド**を使います。書式は次のとおりです。

FILES

```
FILES "0:"  ← ドライブ0のディスクのファイルを表示
FILES "1:"  ← ドライブ1のディスクのファイルを表示
```

- ダブルクォーテーション
- コロン
- ドライブ番号
- ダブルクォーテーション

上のようにキーインしてリターンキーを押せばよいのですが、ドライブ0のディスクのファイルを表示する場合にはドライブ番号の指定を省略して

FILES ⏎

とすることもできます。

私たちは現在システムディスクをドライブ0にセットしてありますから、いま述べた省略形のほうを使い、画面をクリアしてから次のようにキーインしましょう。

```
FILES ⏎
```

画面はどのようになりましたか？

アルファベット、数字が羅列されていて、何がなんだかさっぱりわかりませんね。少しずつ説明しましょう。

なお、標準添付のシステムディスクには「デモンストレーションプログラム」その他、日常あまり使わないものがおさめられています。次ページの画面は、筆者が使い勝手がよいように作り変えたディスクの内容です。説明上は支障ありませんので、このまま読み進んでください。みなさんは本章を読み終えてから、これと同じディスクを作る

〈注〉FILESコマンドを実行すると、上のほうがどんどんスクロールアップして消えてしまいますから、そんなときは ESC キーを押してください。一時的に画面が停止します。表示を再開したければ何かほかのキーを押してください。

とよいでしょう。（5・3節参照のこと）

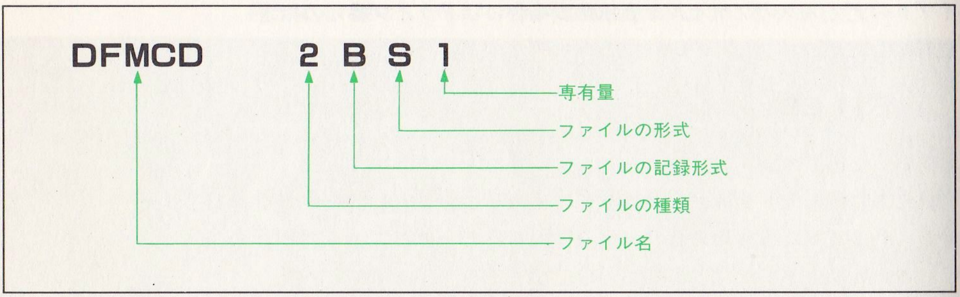
```
FILES

DFMCD      2 B S 1
MCOPY      2 B S 1
SYSDSK     0 B S 1
VOLCOPY    0 B S 1
AUTOUTY    0 B S 1
PFDEF      0 B S 2
FCOPY      0 B S 1
FCOPYM     0 B S 3
FCOPYEB    2 B S 1

140 Clusters Free

Ready
■
```

例として、上の画面の「ＤＦＭＣＤ」の行を取り上げましょう。
１つの行は次のような構成になっています。



上の例にそって簡単に説明しておきますが、まだ理解できない部分も多いと思います。そんな方は、無理に理解しようとせず、「そのうちわかってくるだろう」くらいの軽い気持で先へ進んでください。この本を読み進むうちに、できるだけこれらのことが氷解するように心がけるつもりです。

ファイル名

❶ ファイル名

これは説明がなくともわかると思います。プログラムファイル、データファイルの名前がこれに相当します。

ファイル名の長さは８文字以下と決められています。また、大文

字、小文字は識別されますので、こういった制限の中でわかりやすい名前をつけておくとよいでしょう。

❷ ファイルの種類　　　　**ファイルの種類**

画面に表示される数字によって、私たちはファイルの種類を知ることができます。分類は次のとおりです。

0 …… ベーシックで組んだプログラムファイル
1 …… データファイル
2 …… 機械語で作ったプログラムファイル

❸ ファイルの記録形式　　　　**ファイルの記録形式**

アスキー形式

バイナリ形式

これは、そのファイルがアスキー形式かバイナリ形式かを示すもので、分類は次のとおりです。

A …… アスキー形式
B …… バイナリ形式

プログラムファイルを例にとりますと、**アスキー形式**とは「プログラムの１字１字をそのまま忠実に記録していく(書き込んでいく)形式」です。

たとえば行番号100というのがあったとしますと、「1」のアスキーコードは16進数の「３１」、「０」のアスキーコードは「３０」ですからディスクにイメージ的には

31　30　30

のように記録していきます。

一方**バイナリ形式**とは「プログラムを圧縮して書き込んでいく形式」です。

いまの例でいくと、行番号100は16進になおすと「&Ｈ６４」ですから、単に

64

とだけ記録されます。

このように、アスキー形式とバイナリ形式では記録のされ方が違う上に、ディスク専有量にもだいぶ差がでてきます。ディスクの使用効率からいくとバイナリ形式のほうがずっと有利ですが、アスキー形式のプログラムファイルでないと使えないコマンド、ステートメントもありますので、そのつど使い分けるとよいでしょう。

ファイルの形式

❹ **ファイルの形式**

分類は次のとおりです。

S …… **シーケンシャルファイル**
R …… **ランダムファイル**

シーケンシャルファイル、ランダムファイルについてはあとでくわしく説明することになりますので名称だけ覚えておいてください。

専有量

❺ **専有量**

ディスクもコンピュータ本体と同じように、ある一定の記憶容量をもっていますが、専有量とは、そのファイルがディスク上でどのくらいの容量を使用しているかを示すものです。

単位は**クラスタ**ですが、これもあとで説明します。

各ファイルの役割

14ページの画面をごらんください。いろいろなファイル名が表示されていますが、おもなものについてその役割を簡単に説明しておきましょう。

ユーティリティプログラム

バックアップディスク

❶ **DFMCD**——❸の「SYSDSK」とともに用いるマシン語プログラムです。
❷ **MCOPY**——❹の「VOLCOPY」とともに用いるマシン語プログラムです。
❸ **SYSDSK**——新しいディスクを「システムディスク」にするための**ユーティリティプログラム**です。「ユーティリティ」とは「便利で実用的な」くらいに考えておけばよいでしょう。
❹ **VOLCOPY**——これは、あるディスクの内容を新しいディスクにそっくりコピーするためのプログラムファイルです。コピーしてできあがった新しいディスクのことを**バックアップディスク**といいます。これもユーティリティプログラムです。
❺ **AUTOUTY**——ディスクを自動的に立ち上げるためのプログラムです。
❻ **PFDEF**——PFキーの定義用プログラムです。
❼ **FCOPY**——プログラムファイルまたはデータファイルを他のディスクにコピーするためのプログラムです。

なお、そのほかのユーティリティプログラムについては『ユーティリティプログラム操作手順』をごらんください。

1·4 プログラムのロード

プログラムファイルをディスクから本体のメモリ上に呼びだすことを「プログラムをロード」するといい、ＬＯＡＤコマンドを使います。 ＬＯＡＤ

書式は次のとおりです。

LOAD "ファイルディスクリプタ"

ファイルディスクリプタとは「ドライブ番号とファイル名をコロンで連結した文字列」のことで、たとえばドライブ０のディスクに「ＤＥＭＯ」というファイルがあったとしたら、そのファイルディスクリプタは、 ファイルディスクリプタ

0：DEMO ――――❶

と表わせます。

また、ドライブ１のディスクに「ＤＥＭＯ」というファイルがあったとしたら、そのファイルディスクリプタは、

1：DEMO ――――❷

となります。

ただし、ドライブ０のディスクのファイルに対してのみドライブ番号の指定を省略することができます。つまり、❶の場合ドライブ番号を省略して、単に

DEMO

としても同じことです。

またファイル名は大文字、小文字の区別がありますのでＦＩＬＥＳコマンドで画面に表示されたとおりの文字を使用するようにしてください。もし、「ＤＥＭＯ」を小文字にしてロードしようとした場合、そのファイルが実際にディスクになければ次のようなエラーメッセージが表示されます。

```
LOAD "demo"

File Not Found
Ready
█
```

「File Not Found」、つまり「ロードしようとしたファイル名が見つかりません」というわけです。

では実際にプログラムをロードしてみましょう。

ロードするプログラムファイルは、次節で使用する

SYSDSK

とします。システムディスクを作るためのプログラムです。

次のようにキーインしてください。

```
LOAD "SYSDSK" ⏎
```

〈注〉ファンクションキーの(PF6)に「LOAD"」が定義されていますから、これを使うと便利です。

「Ready」の表示がでましたらロード終了です。2、3秒でプログラムがメモリ上に転送されます。ずいぶん速いですね。

それでは、実際ロードされたかどうか確認してみましょう。

(PF2)キーに「LIST ⏎」が定義されていますから、これを押してリストを画面に表示させてみてください。

次節へ進む前にもう1つだけ勉強しておきましょう。それは、**プログラムをロードしてからRUN（実行）させる方法**です。

私たちがいま学んだ方法では、一度プログラムをロードして、次に「RUN ⏎」を実行するか、あるいは(PF3)キーを押して実行するかしなければいけません。

Rオプション付きロード

ところが、これを一挙にやってしまう方法があるのです。1つはRオプション付きのロードで、書式は次のとおりです。

```
LOAD "ファイルディスクリプタ", R
```

「R」は「RUN」の頭文字で、大文字でも小文字でもかまいません。**「ロードしてからプログラムをRUNせよ」**ということですね。

RUN

もう1つは、直接RUNコマンドを使ってしまう方法です。書式は次のとおりです。

```
RUN "ファイルディスクリプタ"
```

「ファイルディスクリプタで示されるドライブのディスクの中の、目的のファイルをRUNせよ」ということです。これが一番楽そうですね。

1・5 システムディスクのコピーを作る

システムディスクは大切なものです。何らかの手違いでこわれることもありえます。万一の場合にそなえて、いまみなさんが使っているシステムディスクとまったく同じものを作っておきましょう。

ディスクにはドライブ数が1つのタイプ、2つのタイプと2種類ありますが、ここでは2ドライブの場合を例にとって、その手順を説明します。1ドライブのディスクをご使用の方は、この記事を読み、そのあとで『ディスクユーティリティ操作手順』を見ながら実行してみてください。少し手間がかかるだけで、やり方はあまり変わりません。

新しいディスクのセット

まず、新しいディスクを用意してください。

システムディスクは現在「ドライブ0」にセットしてありますね。

次に、この新しい未使用のディスクを「ドライブ1」にセットしてください。

これで準備オーケーです。では、さっそく始めます。

ユーティリティプログラム「SYSDSK」をロードしてRUNさせてください。前節ですでにロードずみの方は PF3 キーを押して実行させてください。

```
RUN "SYSDSK" ↵
```

すると画面に次のメッセージが表示されます。

```
*** FLOPPY DISK FORMATTING ***

DISK FORMATTING ? ■
```

これは「フォーマッティングをしますか？」とたずねているのです。私たちが購入した新しいディスクは、このままではまだ使うことができません。未使用のディスクに「ある一定の情報」を書き込んで初期化しなければなりません。この作業をフォーマッティングといいます。

フォーマッティング

フォーマッティングが必要ですので Y ↵ とキーインします。

すると次のように「(フォーマッティングしようとするディスクのセットしてある) ドライブ番号は何番ですか？」とたずねてきます。

```
WHAT DISK DRIVE(0-3)? ■
```

私たちは「ドライブ1」にセットしてあるので [1] [↵] ですね。

すると次のように、「ドライブ1にフォーマッティングするディスクをセットするよう」求めてきます。そして、「READY？」、つまり「準備できましたか？」と確認しています。

```
SET THE FORMATTING DISK ON DRIVE 1
READY ? ■
```

OKなら [Y] [↵] とキーをたたいてください。するとフォーマッティング作業が開始され、しばらくして画面は次のようになります。

```
DISK FORMATTING COMPLETED !!

*** DISK BASIC SYSTEM COPY ***

WHAT DESTINATION DISK DRIVE(0-3)? ■
```

「フォーマットは完了しました。次にディスクベーシックをコピーします。コピー先のドライブ番号は？」と聞いてきます。私たちは1番ドライブのディスクにコピーしますので [1] [↵] とキーインします。

すると次のように「システムディスクをドライブ0に、システムの転送先のディスクをドライブ1にセットしてください」と問いかけてきます。

```
SET THE SYSTEM DISK ON DRIVE 0
SET THE DESTINATION DISK ON DRIVE 1
READY? ■
```

[Y] [↵] とキーインすると、次のような表示がでてコピー終了です。

```
SYSTEM CODE COPY COMPLETED !!

Ready
■
```

DSKINI

最後にＤＳＫＩＮＩ（ディスクイニシャライズ）コマンドを実行します。次のようにキーインしてください。

```
DSKINI 1 ↵
```

ディレクトリ

ドライブ１のディスクのディレクトリを初期化するということです。ディレクトリとは「ファイルを管理するファイル原簿」のようなものです。リターンキーを押すと「Are you sure (Y or N) ?」の確認メッセージが表示されますからＯＫなら「Y」をキーインしてください。

これで、新しいディスクのフォーマッティング、ディスクコードのコピーが終了しましたので、**この新しいディスクは「システムディスク」となりました。**システムディスクですから、このディスクによってディスクシステムを起動させることができます。

この「新しいシステムディスク」は、まだディスクコードをコピーしただけですから、オリジナルの「システムディスク」に書き込まれているユーティリティプログラムとかデモンストレーションプログラムはコピーされていません。

次のようにキーインして調べてみましょう。

```
FILES "1:" ↵
```

次のような画面になりましたね。

```
FILES "1:"
          ←――――――――――――――――――――ファイル名なし
152 Clusters Free

Ready
█
```

ファイル名は１つも表示されませんね。

そして**「152クラスタ」**が私たちが使うことができる容量です。

新しいシステムディスクで立ち上げる

それでは、この新しいディスクでシステムを起動させることができるかどうかテストしてみましょう。

次の手順で行ってください。

1. **ドライブ0、ドライブ1からディスクを取り出します。**
2. **「新しいシステムディスク」を「ドライブ0」にセットします。**
3. **本体前面のリセットボタンを押します。**

これから先は1・2節で説明した手順と同じです。12ページのようなオープニングメッセージが表示されたと思います。

ファイルをすべてコピーする

さて、新しいディスクを購入して容量を最大限に使いたいという場合には、ディスクコードのコピーまでで終りにしてよいのですが、オリジナルのシステムディスクとまったく同じ内容のディスクをもう1つ作っておきたいものです。

それでは作業を始めますが、これは「VOLCOPY」というユーティリティプログラムを実行することによって行ないます。

ディスクは立ち上げてありますね？

そうしたら、**オリジナルのシステムディスクをドライブ0に、ディスクコードだけをコピーした新しいディスクをドライブ1にセット**してください。さあ、次の順序で作業開始です。

「VOLCOPY」をRUNさせてください。

```
RUN "VOLCOPY" ⏎
```

すると画面に次のメッセージが表示されます。

```
*** DISK COPY ***

WHAT SOURCE DISK DRIVE(0-3)? ■
```

「ソースディスク（元になるディスク）がセットしてあるドライブは何番ですか」とたずねています。

私たちはドライブ0にセットしてあるので [0] ⏎ とキーインします。すると次に**「デスティネーションディスク（コピー先のディスク）のセットしてあるドライブ番号」**をたずねてきます。

```
WHAT DESTINATION DISK DRIVE(0-3)? ■
```

私たちはドライブ1にセットしてありますので、[1] ⏎ とキーイン

します。

1 ⏎

すると今度は次の画面表示となります。

```
SET THE SOURCE DISK ON DRIVE 0
SET THE DESTINATION DISK ON DRIVE 1
READY? ■
```

「ソースディスクをドライブ0に、デスティネーションディスクをドライブ1にセットしてください」ということです。私たちはすでにセットずみですから確認の意味と考えればよいでしょう。

「Y」をキーインしましょう。

Y ⏎

これからいよいよファイルのコピーです。画面には次のように表示されます。

```
TRACK = 0 -- 4
TRACK = 5 -- 9
TRACK = 10 -- 14
```

これはどういうことかといいますと、音楽のレコード盤を思い浮かべてください。幾重もの輪で構成されていますね。ディスクも同じようになっていて、一番外側からトラック0、トラック1……というように番号がつけられています。つまり、上の画面はコピーされているトラック番号が表示されているのです。

トラック

意味がよくわからない方はそのままにして先へ進んでください。

このようにトラックごとにコピーを行い、終了すると次のような画面になります。

```
TRACK = 30 -- 34
TRACK = 35 -- 39
COPY FINISHED!

CONTINUE VOLUME COPY(Y/N)? ■
```

「ファイルのコピーは終りました! ボリュームコピー(すべてのファイルのコピー)を続けますか?」とたずねています。

つまり、ほかのディスクにシステムディスクの中のファイルをコピーする作業を続けますか、と聞いているのです。私たちはこれで終りにしますので N ⏎ とキーインしてください。「N」は「NO」の頭文字です。

これで、すべての作業が終りました。

では、新しいディスクにファイルがコピーされているかどうか調べてみましょう。

まず、「ドライブ0」側のシステムディスクをもう一度画面に表示してみましょう。次のようにキーインすればよかったですね。

```
FILES ⏎
```

13ページの場合と同じように、いろいろなファイル名が表示されましたね。

では次に、「ドライブ1」側のディスクの内容を表示させてみましょう。今度は次のようにキーインしてください。

```
FILES "1:"
```

ドライブ0のディスクについては「FILES」でよかったのですが、**ドライブ1側のディスクの場合はさらにそのうしろに「"1:"」をつけ加えます。**

さて、どうですか？ ドライブ0のシステムディスクとまったく同じ内容が表示されたと思います。

これで本節は終りですが、**「オリジナルのシステムディスク」**は大切に保管しておいてください。

次に2枚目の**「未使用のディスク」**を用意し、「コピー版のシステムディスク」を使って「ディスクコードのコピー」を行い（19～21ページの操作）、**この新しいディスクに対して5・3節の操作を実行**してください。**本書では、このディスクを「ドライブ0」にセットして使うことにします。**

少々混乱しそうですが、整理すると次のようになります。

❶ オリジナルのシステムディスク――保管する

❷ コピー版のシステムディスク――保管する

❸ 5・3節を実行して作ったシステムディスク――ドライブ0にセットする

❹ 未使用のディスク――次章で使います

1・6 プログラムのセーブ

さて次は、プログラムのセーブです。

15ページをごらんください。ファイルの記録形式には2通りあります。1つが**アスキー形式**、もう1つは**バイナリ形式**（非アスキー形式）です。

アスキー形式
バイナリ形式

この2つの記録形式はセーブ方法によって区別されますが、前にも説明したように「アスキー形式」は「プログラムの1字1字をそのままの形でディスクに書き込み」、「バイナリ形式」は「圧縮した形で書き込み」ます。

この節では、2つともテストしてみることにしましょう。

サンプルプログラムの入力

では、「プログラムのセーブ」を学習するためのサンプルプログラムをキーインしましょう。

すでに何かプログラムが入力してある方は、**NEWコマンド**でメモリをクリアしてから入力してください。

NEW

また、前ページで作った❸のシステムディスクがドライブ0にセットしてあることを確認してください。

プログラムを次に示します。

```
100 ' demonstration program
110 ' save "demoprog"
120 '
130 WIDTH 40,25 : CONSOLE 0,25,0,0
140 CLS
150 '
160 LINE (10,10)-(630,190),PSET,6,B
170 LINE (150,30)-(490,60),PSET,6,B
180 PAINT (50,15),5,6
190 '
200 LOCATE 13,5 : COLOR 4
210 PRINT "FM-77 ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ"
220 END
```

別に深い意味のあるプログラムではありません。ＬＩＮＥ文で画面に長方形を2つ描き、ＰＡＩＮＴ文でその一部を塗りつぶし、窓にな

ったところに「FM－77　プログラミング」と表示するだけです。

バイナリ形式のセーブ

バイナリ形式

SAVE

入力できましたか？

入力できたら、いよいよこのプログラムのセーブです。

プログラムをディスクに書き込むには「**ＳＡＶＥ（セーブ）コマンド**」を使います。書式は次のとおりです。

```
SAVE "ファイルディスクリプタ"
```

この書式に準じて行なう記録方法が「バイナリ形式」です。

プログラム名は仮りに「demoprog」としておきましょう。

次のようにキーインしましょう。

```
SAVE "0:demoprog" ⏎
```

〈注〉ファンクションキーの(PF7)に「SAVE"」が定義されていますから、これを使うと便利です。

「ＳＡＶＥ」は大文字でも小文字でもかまいませんが、ファイル名は小文字にしてください。

リターンキーを押すとアクセス表示用ＬＥＤが赤く点灯し、まもなく「Ready」が表示されてセーブが終了します。

```
SAVE "0:demoprog"

Ready
■
```

では、実際にセーブされたかどうかチェックしてみましょう。

ＦＩＬＥＳコマンドを使えばよいですね。

```
demoprog   0 B S 1

139 Clusters Free

Ready
■
```

上の画面は、表示されたファイル名の一番下の行だけを示してあります。

画面を見るとサンプルプログラムが確かにセーブされていますね。

そして、私たちが使うことのできる残りの容量は139クラスタです。
次をごらんください。

```
demoprog   0  B  S  1
```

- 1 → 1クラスタ使用
- S → シーケンシャルファイル
- B → バイナリ形式
- 0 → ベーシックプログラム

確かにバイナリ形式でセーブされていますね。

アスキー形式のセーブ

アスキー形式

それでは次に、まったく同じプログラムをアスキー形式でセーブしてみましょう。

アスキーセーブの書式は次のとおりです。

```
SAVE "ファイルディスクリプタ", A
```

バイナリ形式と異なる点は、うしろに「, A」が追加されたことです。「A」は大文字でも小文字でもかまいません。

このようにアスキーセーブの場合、うしろに「A」をつけることから「**Aオプション付セーブ**」ともいいます。

Aオプション付き
セーブ

ではさっそくテストしてみましょう。次のようにキーインします。

```
SAVE "0:demoprog",A ⏎
```

ファイルディスクリプタの「0:」は省略してもかまいません。リターンキーを押すと、次のような**確認メッセージ**が表示されます。

```
Are you sure(Y or N)?
```

あるプログラムは初めてディスクにセーブするときはこのメッセージは表示されませんが、プログラムを修正したりしてもう一度セーブしなおそうとすると表示されます。

これはどういうことかといいますと、たとえば「P」という名前のプログラムがすでにディスクにセーブされていたとします。そして、この「P」というプログラムを手直しして、同じファイル名でセーブしたとすると、修正前のプログラムは消されて修正後の新しいプログラムが上書きされます。

「修正前のプログラムも残しておくのだった」と後悔されることも多々あります。

このような事故を防ぐためにもワンクッションおいて、メッセージを表示して確認をうながしているのです。

私たちは現在はこの心配はありませんから、大文字でも小文字でもいいですから「Y」を入力しましょう。

すると、カチッと音がして間もなくセーブが終了します。

```
SAVE "0:demoprog",A
Are you sure(Y or N)? Y

Ready
█
```

では再びＦＩＬＥＳコマンドでファイル名を表示させてみてください。

```
demoprog  0 A S 1

139 Clusters Free

Ready
█
```

いかがですか？

今度は先ほどの「B」のところが「A」に変化していますね。つまり、無事にアスキー形式でセーブされたのです。

これでプログラムのセーブは終わりです。

それはそうと、サンプルプログラムは正常に動作しますか？

いくらサンプルだとはいえ、うまく走るかどうかテストしておいてください。

1・7 プログラムの連結

２つの相異なるプログラムをつなげて１つのプログラムにすることができたら、ずいぶん便利だと思いませんか？

たとえばメインルーチンを入力し、サブルーチンはすでにディスクにセーブされている別のプログラムのそれを利用することにして、この２つをドッキングして１つにまとめることができたら、ずいぶん手間をはぶくことができます。

この作業をやってくれるのがMERGE（マージ）コマンドです。

MERGE

サブルーチンの入力

では、前節で作ったサンプルプログラムに新しくサブルーチンをつなげることにしましょう。

前節のプログラムがまだメモリ上にある方は、NEWコマンドでクリアしてから次のサブルーチンを入力してください。

〈注〉下のプログラムの1030行に「" "」がありますが、これはダブルクォーテーションを２つ続けてキーインしてください。間にスペースを入れてはいけません。

```
200 GOSUB 1000
210 END
1000 '
1010 TITLE$="FM-77 ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ "
1020 COLOR 4
1030 WHILE I$=""
1040   I$=INKEY$
1050   TITLE$=MID$(TITLE$,2)+LEFT$(TITLE$,1)
1060   FOR I=1 TO 100 : NEXT I
1070   LOCATE 13,5 : PRINT TITLE$
1080 WEND
1090 RETURN
```

キーインが終わりましたら、「mrg」というファイル名でアスキーセーブしてください。バイナリ形式ではダメですよ。

```
SAVE "mrg",A ⏎
```

セーブできたら、前節のプログラムをロードしてください。

```
LOAD "demoprog" ⏎
```

これで、この節で作ったサブルーチンはアスキー形式でディスクに

セーブされ、本体のメモリ上には前節で作ったメイン部分がロードされたことになります。

ＬＩＳＴコマンドで確認してみてください。

では、いまメモリ上にロードされたメイン部分に、ディスクからサブルーチンを呼びだして１つにまとめてみましょう。

ここでＭＥＲＧＥコマンドを使うことになりますが、書式は次のとおりです。

```
MERGE "ファイルディスクリプタ"
```

マージに移る前に１つ注意があります

25ページと29ページのプログラムを見比べてください。どちらにも200行、210行という行番号がありますね。

私たちは、25ページのメインプログラムに29ページのサブルーチンをマージするわけですが、この場合、重複する行番号があると、マージするほうのステートメントが優先されます。

つまりこの場合、メインプログラムの200行、210行はサブルーチン側の200行、210行で置き換えられます。

では実際にやってみましょう。次のようにキーインしてください。

```
MERGE "mrg" ⏎
```

やがて「Ready」が表示されてマージ終了です。

ＬＩＳＴコマンドでプログラムリストを画面に表示させてみてください。うまく連結することができたでしょう。

よく見ると220行のＥＮＤ文が１つ多いので、この行を削除しておきましょう。

```
220 ⏎
```

では、(PF3) キーでプログラムを実行してみてください。

「ＦＭ－７　プログラミング」という見出しが電光掲示板のようにクルクルまわっています。

どのキーでもいいですから押してください。処理が終了します。

さあそれでは、このプログラムをセーブしてマージを終わりにしましょう。

```
SAVE "demoprog" ⏎
```

1・8 プログラムの削除

ディスクを使っているとファイルもだんだんふえてきて、なかには不必要なファイルもでてきます。

そんなときに登場するのが「ＫＩＬＬ（キル）コマンド」です。 KILL

書式は次のとおりです。

KILL "ファイルディスクリプタ"

このコマンドはダイレクトモードでもプログラムモードでも使うことができますが、ダイレクトモードの場合には、本当にＫＩＬＬ（削除）してよいのかどうか確認メッセージが表示されます。

ではさっそくテストしてみますが、先ほどマージ用にセーブしたプログラム「mrg」を削除することにします。私たちは、メインプログラム、サブルーチンをマージして１つにしましたので、「mrg」は必要なくなりました。

まずＦＩＬＥＳコマンドで「mrg」がディスクにセーブされていることをもう一度確認してください。

```
FILES "0:"

DFMCD      2 B S 1
MCOPY      2 B S 1
SYSDSK     0 B S 1
VOLCOPY    0 B S 1
AUTOUTY    0 B S 1
PFDEF      0 B S 2
FCOPY      0 B S 1
FCOPYM     0 B S 3
FCOPYEB    2 B S 1
demoprog   0 B S 1
mrg        2 A S 1 ←―――――― これから削除するファイル

138 Clusters Free

Ready
■
```

確かにセーブされていますね。

それでは次のようにキーインしてください。

```
KILL "0:mrg" ⏎
```

すると、次のように確認メッセージが表示されます。

```
Are you sure(Y or N)?
```

大文字でも小文字でもかまいませんから「Y」をキーインしてください。すぐに「Ready」が表示されて削除終了です。

本当にＫＩＬＬされたかどうか、ＦＩＬＥＳコマンドで確認してみましょう。

```
FILES "0:"

DFMCD       2 B S 1
MCOPY       2 B S 1
SYSDSK      0 B S 1
VOLCOPY     0 B S 1
AUTOUTY     0 B S 1
PFDEF       0 B S 2
FCOPY       0 B S 1
FCOPYM      0 B S 3
FCOPYEB     2 B S 1
demoprog    0 B S 1
                     ←削除されている
139 Clusters Free

Ready
█
```

どうですか？　確かに「mrg」が消えていますね。

この「mrg」には１クラスタ使われていましたから、ＫＩＬＬすることによってフリークラスタは「138」から「139」に１クラスタだけふえています。

ＤＳＫＩＮＩ

なお、すべてのファイルを削除したいときは**ＤＳＫＩＮＩコマンド**を使いますが、全部消されてしまいますから注意してください。

1·9 ファイル名の変更

あるファイル名でプログラムをセーブしたのはいいが、あとで考えるとあまり気に入らない、というケースも生ずると思います。

私たちがこれまで学んだ命令だけでファイル名を変更しようと思うと、こんな手順をふまなければなりません。たとえばファイル名PをQに変更する場合を考えてみます。

❶ **プログラム「P」をロードする**

❷ **ロードしたプログラムを「Q」というファイル名でセーブする**

❸ **KILLコマンドで「P」を削除する**

これで、「P」というファイル名を「Q」に変更することができましたが、ちょっとめんどうですね。これを1プロセスでやってしまうコマンドがあります。**NAME（ネーム）コマンド**です。 **NAME**

書式は次のとおりです。

```
NAME "旧ファイルディスクリプタ" AS "新ファイルディスクリプタ"
```

「古いファイルを新しいファイルとしてネーミングする」ということになります。

では実践してみましょう。「demoprog」を「testprog」というファイル名に変更してみましょう。いずれもドライブ0にセーブしてありますから、ファイルディスクリプタのドライブ番号は省略します。

次のようにキーインしてください。

```
NAME "demoprog" AS "testprog" ⏎
```

それではFILESコマンドで確認してみましょう。

```
testprog  0 B S 1
139 Clusters Free
Ready
█
```

新しいファイル名につけかえられましたね。

1・10　デモプログラムの実行

ここで、本章を終わるに当たってちょっと息ぬきしましょう。

システムディスクに入っている富士通のデモンストレーションプログラムを楽しんでみます。

システムディスクの中の「DEMO」というプログラムを実行すればよいのです。

24ページに述べた❷の「コピー版システムディスク」をドライブ0にセットしてください。

次のようにキーインします。

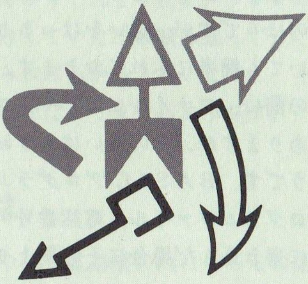
```
RUN "DEMO" ↵
```

さあ、デモンストレーションが始まりました。いかがですか？

このプログラムはエンドレスになっていますから、終わりにしたいときは本体前面のＲＥＳＥＴボタンを押してください。ディスク起動状態に戻ります。

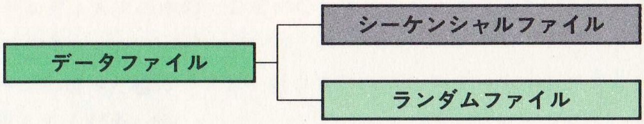
第2章

シーケンシャルファイル[I]



2・1　データファイルについて

情報の集合体のことを**データファイル**といいますが、これはその性質によって次のようにわけることができます。



この章で学習するのはデータファイルのうちのシーケンシャルファイルのほうですが、違いをはっきりさせるために、ランダムファイルについても簡単にふれておきます。

その前に、ファイルにはデータファイルのほかに**プログラムファイル**もありますが、その違いはもうおわかりですね？

そうです、ＢＡＳＩＣプログラムなどをディスクにセーブした場合に**プログラムファイル**、電話番号・住所・氏名といったデータをディスクに書き込んだ場合にその集まりを**データファイル**といいます。

プログラムファイル
データファイル

シーケンシャルファイルもランダムファイルも同じデータファイルなのですが、ディスクに書き込まれる状態が違います。

「シーケンシャル」というのは「連続的な」という意味です。「ランダム」というのは直訳すれば「デタラメナ」ということですが、デタラメというよりは「連続的ではない」くらいに理解しておいたほうがよいようです。

〈注〉ＦＩＬＥＳコマンドでシステムディスクにセーブされているファイル名を画面に表示すると、プログラムファイルも形式は「Ｓ」、すなわち「シーケンシャルファイル」になっていますが、本書での「シーケンシャルファイル」という呼び方は「データファイル」に限定することにします。

シーケンシャルファイルとは

シーケンシャルファイルとは、ディスクに１番目のデータから連続的に書き込まれたファイルのことで、これをディスクから読み出そうとするときに、書き込まれた順序でしか読み出せません。

たとえば電話帳のファイルをシーケンシャルで作った場合、夏目さんという人の電話番号を知りたくても、もし50音順に書き込まれていれば、安藤さん、伊藤さん……と、夏目さんの前に登録してある人のデータを読み出していかなければなりません。

あなたは、カセットテープを使ってファイルを作りましたか？　あれがシーケンシャルファイルです。

シーケンシャルファイルの特徴をまとめると次のとおりです。

❶　データをロード・セーブする場合、必ずファイルの先頭から順次行います。
❷　ファイル内の一部だけを書き換えることができません。
❸　ファイルを変更できるのは、ファイルの最後に新しいデータを追加することだけです。
❹　プログラム面、あるいはファイル設計の面から見ても、ランダムファイルに比較して簡単です。
❺　先ほどの電話帳の例でいいますと、夏目さんの電話番号・住所・氏名を**1レコード**（このレコードの集まりがファイルです）と呼びますが、この大きさは255文字以内でしたら自由に変えられます。　**レコード**

意味があまりわからない人もいるかもしれませんが、「ランダムファイルより扱いが簡単だ」くらいに考えていただき読み進んでください。あとで実際にプログラムを組んでみればわかると思いますし、そのときになってもう一度読みなおしていただいてもけっこうです。

ランダムファイルとは

次に**ランダムファイル**ですが、やはり電話帳の例にそってお話しします。　**ランダムファイル**

電話帳ファイルをランダムファイルとしてディスク上に作った場合、たとえば夏目さんにかかわるデータを知りたいとき、ほかの関係ないデータまで読み出す必要はなく、直接に取り出すことができます。したがって、時間的なロスもなく大変効率的です。

ランダムファイルの特徴も列挙しておきましょう。

❶　ファイル内の各レコードを任意に読み書きできますので、処理時間を短縮できます。
❷　ファイル内の一部のデータだけを書き換えることができます。
❸　データの追加も自由です。
❹　シーケンシャルファイルに比べてプログラムが複雑になります。
❺　1つ1つのレコードの大きさは256バイト以内の一定バイト数に固定されています。

シーケンシャル、ランダムとも、あまりよく理解できない方もいるかもしれませんが、実践していくうちにのみこめてくると思いますので、安心して先へお進みください。

2·2　ディスクの構造

この節では、「3.5インチマイクロフロッピィディスク」の構造について説明しますが、まだ実際にプログラムによるファイル操作をていませんので少しわかりにくいかもしれません。

とくに、ディスクの構造がどうなっているかということは、プログラミングする上であまり大切ではありません。ですから、ちょっととまどいを感じる人は、この節をとばして次へ進んでいただいてもかまいません。

まず「ディスク」の図を右ページに示しますので、これを見ながらお読みください。

「3.5インチマイクロフロッピィディスク」とは、「3.5インチ両面倍密度ディスク」とも呼ばれ、中に入っている円盤状の磁性体の直径が「3.5インチ」で、「両面」つまり「表面」と「裏面」の密度が倍のディスクということです。右図の1つのセクタの記憶容量が256バイトのものを倍密度、128バイトのものを単密度といいます。

図を見るとわかるように「ディスク」は、セクタ、トラック、サイド（これは表か裏かということで、表をサイド0、裏をサイド1といいます）で構成され、その仕様は次のようになっています。

トラック数

❶ トラック数——片面あたりトラック0～39の40トラックがあり、表裏があるので総トラック数は80になります。

セクタ数

❷ セクタ数——1つのトラックあたりセクタ1～16の16セクタがあり、片面の総セクタ数は16×40＝640セクタ、したがって両面では640×2＝1280セクタということになります。

セクタ長

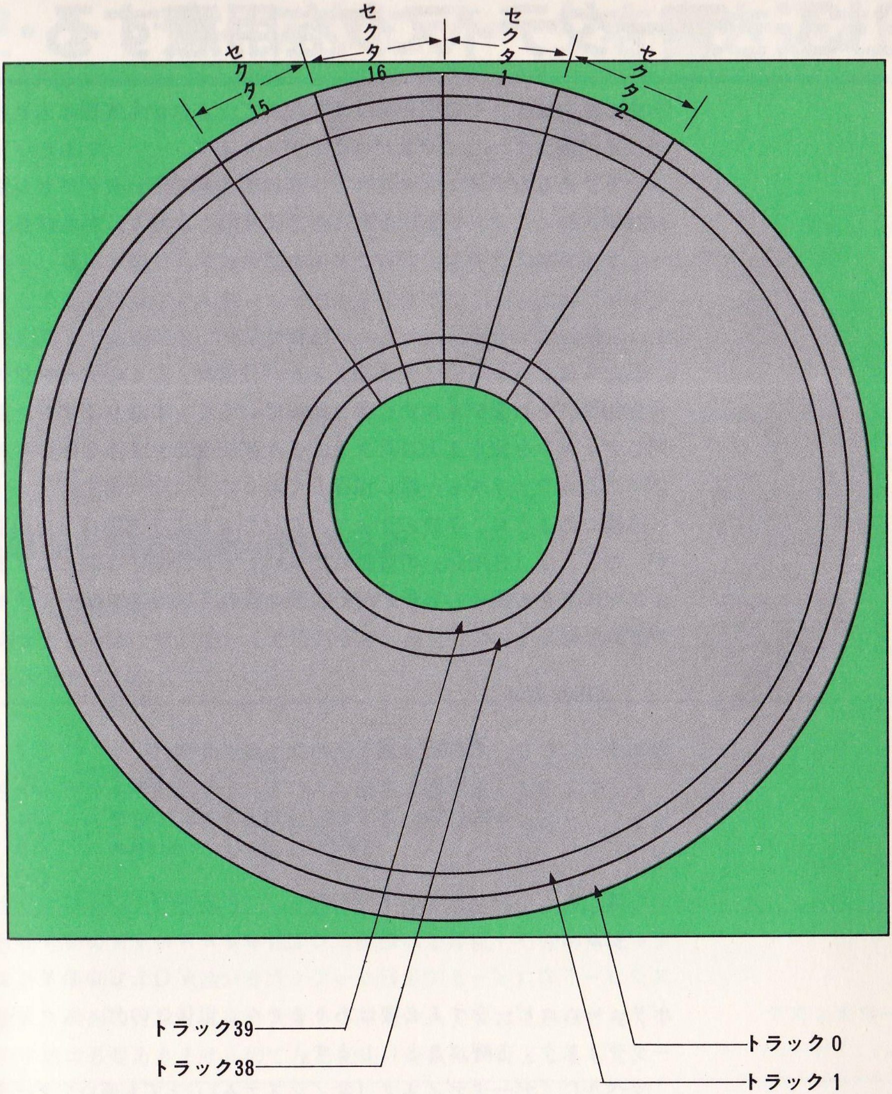
❸ セクタ長——1セクタが256バイトです。

ちょっと理解しにくいかもしれませんが、ディスクというのはイメージ的にはレコード盤のようなものだと考えてください。徐々におわかりいただけると思います。

クラスタ

なお、データファイルにしろプログラムファイルにしろ、ディスクに書き込まれるときは「クラスタ」単位で行なわれます。

セクタが8つ集まったものを「1クラスタ」と呼びます。ですから、右ページの図で、トラック0の「セクタ1～セクタ8」までが1つのクラスタ、「セクタ9～セクタ16」も、やはり1つのクラスタという



ことになります。

1トラック＝2クラスタ

となりますから、ディスク全体では

（2クラスタ）×（40トラック）×（2サイド）＝160クラスタ

になります。

2・3　データディスクを用意する

さて、おおまかな説明が終りましたので、これから実践に入りますが、その前に１つしておきたいことがあります。

いま、あなたのディスクユニットには第１章で作った**「コピー版のシステムディスク」**がセットしてありますね。そしてこれにはユーティリティプログラムなどがおさめられています。

私たちはこれから、「英単語」とか「住所録」などのデータファイルを作り、それをディスクに書き込んでいくことになりますが、もし同じディスクを使うことになりますと、データファイルを作るためのプログラムファイルも一緒に保存しておくことになります。

当面このように、１枚のディスクだけでもあまり支障はないのですが、たとえば「住所録」の登録数が500件とか1000件にふえてきますと、ディスクの残りの容量がだいぶ狭められてきて、フリークラスタがゼロになると、

Disk Full

のエラーになり、それ以上書き込めなくなります。

そこでこのような事態も考慮に入れて、もう１枚新しいディスクを用意し、これを**「データファイル専用のディスク」**としたいと思います。

では、**新しいディスクを１枚用意し、これをドライブ１にセットして、19から21ページまでの処理、つまりフォーマッティングからディスクコードのコピーまでを行なってください。**ＶＯＬＣＯＰＹによるボリュームコピーをする必要はありません。以後このディスクを**「データディスク」**と呼ぶことにします。

ラベルに「データディスク（含／システム）」とでも書いてデータディスクに貼っておいてください。

データディスク

〈注〉ここで作ったデータディスクも、システムのコピーをしていますから「システムディスク」になりますが、「データだけを記録する」という意味で「データディスク」という表現を使います。

では次節からプログラミングに入りますが、ディスクのセットは次のように行なってください。

システムディスク　➡　ドライブ0

データディスク　➡　ドライブ1

2·4 ディスクにデータを書き込む

さて、おまたせしました。これからいよいよデータの読み書きを学んでいきます。

ここで取り上げるプログラムは**「英単語練習プログラム」**です。しかしこの本では、英語の勉強が目的ではありませんから、扱う単語もやさしい基本単語にしておきます。

もしあなたにお子さんがいらっしゃるようでしたら、このプログラムを応用して本格的な「英単語練習プログラム」を作ったら楽しいと思います。

とにかく、この本ではデータの扱い方、プログラミングの基本をマスターしていただくことに重点をおいてお話しします。

〈注〉たとえば、LISTコマンドで40行を表示させ、カーソルを「4」のところへ移動し、「SAVE」の「S」がカーソルのところへくるまで DEL キーをたたきます。「S」がカーソルに重ったらリターンキーを押してください。このプログラムがディスクにセーブされます。

英単語と日本語訳の書き込み

プログラムの前のほうには**REM文**を入れて、何をするプログラムなのかわかるようにしておくと便利です。**NEWコマンド**を実行してから入力してください。

```
10 ' file name ==> EITAN1
20 ' 1984.6.10
30 ' sequential data write
40 ' save "0:EITAN1"
50 '
```

行番号40のステートメントは、プログラムを手直ししたのちセーブするとき便利です。〈注〉

次に、画面の行数と文字の大きさですが、入力にはそれほどたくさんの字数は必要ありませんので左右40桁、行数は20行としました。

```
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
```

シーケンシャルのデータファイルに数値とか文字列を書き込むには**PRINT#文**（プリントシャープ、またはプリントナンバー）を使い、書式は次のとおりです。

PRINT#

```
PRINT #(ファイル番号),(出力ならび)
```

ファイル番号

「出力ならび」には「数値」あるいは「文字列」が入ります。

また「ファイル番号」には0から15までの整数を指定できますが、システムディスクを起動させたときに画面に表示された

```
How many disk files(0-15)?
```

というメッセージがありましたが、このとき入力した数値をこえてはいけません。「同時にいくつファイルを使いますか？」というあなたへの問いかけでしたね。

それではさっそく書き込み部分のプログラムを作ってみましょう。次のようになります。

```
120 INPUT "ｴｲｺﾞ       ";A$
130 INPUT "     ﾆﾎﾝｺﾞ";B$
140 PRINT #1,A$;",";B$
```

- 120・130行：英単語と日本語の入力
- 140行：ディスクへデータを書き込む
- ",": ダブルクォーテーションでコンマをはさむ

さあ、RUNさせてみてください。適当な英単語と日本語訳を入力してみましょう。

おや？　ピーッというブザー音とともにエラーメッセージが返ってきました。精神衛生上よくない音ですが、まあ我慢しましょう。

次のような画面になってコマンド待ちになってしまいました。

```
ｴｲｺﾞ       ? airplane
     ﾆﾎﾝｺﾞ? ﾋｺｰｷ

File Not Open In 140
Ready
█
```

- ｴｲｺﾞ／ﾆﾎﾝｺﾞ：どちらも入力後リターンキーを押す
- File Not Open In 140：エラーメッセージ

直訳すると「140行でファイルがＯＰＥＮされていない」ということです。ファイルが開かれていないとはどういうことでしょうか。

実はファイルというのは、たとえてみればノートのようなもので、開いてなければ書き込みができないのです。

あなたがノートに何か書くとき、開いてから字を書きますね。ディスクの読み書きも原理はこれとまったく同じなのです。

ここで再び、シーケンシャルファイルとランダムファイルの違いについてふれておきます。

ディスクをノートにたとえますと、シーケンシャルファイルの場合、１ページ目から順序よく書いていかなければならないのです。一方、ランダムファイルの場合は、自分の好きなページに、つまり、10ペー

ジ目でも自由に書き込むことができるのです。

また、すでに書き込んであるノートのあるページの文章だけ読みたくなったとき、シーケンシャルファイルですと、自分の意志とは関係なく、１ページ目から、１、２、３、……と順序よくページをめくっていかなければならないのです。

ところが、ランダムファイルの場合、ノートにインデックスがついているようなもので、直接自分の希望するページを開いて読むことができるのです。ずいぶん便利ですね。

これで２つのファイルの違いがイメージ的に以前よりはっきりしてきたのではないかと思います。

ファイルのOPENとCLOSE

さて、画面にエラーメッセージが出たままストップしてしまいましたが、いまのたとえ話でもおわかりいただけたように、ファイルという名のノートをまずＯＰＥＮする、つまり開かなくてはいけないのです。そして、これも大切なことですが、書き終えたら閉じておく、つまりＣＬＯＳＥしなければならないのです。

OPEN

CLOSE

ＯＰＥＮ文は常にＣＬＯＳＥ文と対で使われます。

書式は次のとおりです。

OPEN 　"モード"，＃（ファイル番号），"ファイルディスクリプタ"
CLOSE ＃（ファイル番号），＃（ファイル番号），……

モードについて

モードはモード文字**「Ｉ」「Ｏ」「Ａ」または「Ｒ」**によって指定しますが、**「Ｒ」はランダムファイルの場合**ですので、ここでは省略します。「Ｉ」「Ｏ」「Ａ」はそれぞれ入出力の違いを示しますが、その意味は次のとおりです。

モード

❶ **Ｉ（インプットモード）**――指定されたファイルからデータを読み出すときに使います。ファイルがディスク上にないとエラーとなります。

インプットモード（INPUT MODE）

❷ **Ｏ（アウトプットモード）**――指定されたファイルにデータを書き込むときに使います。ファイルディスクリプタで示されるファイル名がすでにディスク上に存在する場合にはエラーとなります。

アウトプットモード（OUTPUT MODE）

❸ **Ａ（アペンドモード）**――すでにディスクに書き込まれているシ

アペンドモード（APPEND MODE）

ーケンシャルファイルの末尾にデータを追加したいときに使います。ファイルディスクリプタで示されるドライブに、指定されたファイル名がないとエラーになります。

「ファイル番号」はPRINT＃文の場合と同じです。また、「ファイルディスクリプタ」もLOAD、SAVEのときと同じですからわかりましたね。

ところでデータファイルの場合、それとはっきりわかるようにファイル名の一部に「DAT」などを入れておくと便利です。「DATA」の頭3文字をとったわけです。こうしておけばすぐデータファイルだということがわかりますね。

そこで私たちは、英単語および日本語訳のためのデータファイルに

EITANDAT

と命名することにします。

ファイル番号は140行のPRINT＃文で「1」を使っていますから、シーケンシャルファイルの3つのモードは次のように表わすことができます。

```
OPEN "I", #1,"1:EITANDAT"   ←インプットモード
OPEN "O", #1,"1:EITANDAT"   ←アウトプットモード
OPEN "A", #1,"1:EITANDAT"   ←アペントモード
```

〈注〉1ドライブタイプをご使用の方は、115行のファイルディスクリプタのドライブ番号指定を「0：」にしてください。以後、ドライブ番号指定については同じ扱いになります。

さて、私たちは現在どのモードを必要としているでしょうか？

そうです。英単語のファイルを新しく作ってデータを書き込むのですから「アウトプットモード」ですね。

では、これを使って書き込み部分をキーインしておきましょう。

先ほど述べたように**OPENしたら必ずCLOSEしなければなりません**から、ついでにCLOSE文も入力しておきます。

```
115 OPEN "O",#1,"1:EITANDAT"
150 CLOSE #1
160 END
```

上のプログラムでは160行としてEND文を入れています。END文を入れておくと、**すでにOPENされているファイルはすべてCLOSEされます**ので、150行でわざわざCLOSE文を入れる必要はないのですが、一応行儀よく書式にのっとって話を進めます。

また、ＣＬＯＳＥ文のうしろのファイル番号を省略しますとやはり、すでにＯＰＥＮされているすべてのファイルがＣＬＯＳＥされますから注意してください。

さあ今度はエラーが出ずに実行できるでしょうか。PF3 キーで走らせてみてください。

```
ｴｲｺﾞ        ? airplane
      ﾆﾎﾝｺﾞ? ﾋｺｰｷ


Ready
■
```

確かに成功したようですが、１単語登録しただけで終ってしまいました。もう少しデータ数をふやすことはできないでしょうか。

120行のステートメントにプロンプト文を入れ、さらに「99」を入力したらＥＮＤ処理へ飛び、その他の場合はデータを書き込んだのちに120行へいくようにしてみましょう。

次のステートメントを追加します。

```
120 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
125 IF A$="99" THEN GOTO 150
145 GOTO 120
```

プロント文を入れましたので、120行および130行のＩＮＰＵＴ文によるクエスチョンマーク「？」の出力位置がズレてきますから、130行のプロンプト文の「ニホンゴ」の前にスペースを３つインサートしておきましょう。

```
130 INPUT "            ﾆﾎﾝｺﾞ";B$
```

さあ、これでＲＵＮしてみてください。

おやっ、またエラーメッセージが表示されてしまいました。

```
File Already Exists In 115
Ready
■
```

このエラーの意味はこれから説明しますが、その前にダイレクトモードで次をキーインしておいてください。

```
CLOSE ↵
```

このダイレクトステートメントをなぜ行なうかといいますと、これから先、プログラムを実行してテストしたりするケースが多くなりますが、先ほどのようにエラーメッセージがでてプログラムが停止してしまった場合、ファイルがOPENされたままである可能性があるからです。

くり返しますが、OPENしたファイルは用がすんだらCLOSEしておく、「本を読み終えたら、ちゃんと閉じておかなければなりませんね」、これと同じです。

ファイルをOPENしたままディスクをディスクユニットから抜き出したりしますと、ディスクがこわれてしまう場合もありますから注意してください。

それではエラーメッセージの説明に戻ります。

File Already Exists In 115

意味は「115行で指定されたファイルはすでに（ディスク）にありますよ」ということです。

43ページのモードの解説の❷のところをごらんください。アウトプットモードでファイルをOPENする場合、指定したファイルと同一のファイルがディスクに存在してはいけないのでした。

しかたないですから、ファイルをKILLしてからプログラムを実行してみることにしましょう。

```
KILL "1:EITANDAT" ⏎
```

それではRUNしてみてください。今度はうまくいくと思います。

```
エイゴ゛（オワリ=99)? airplane
      ニホンゴ゛? ヒコーキ
エイゴ゛（オワリ=99)? apple
      ニホンゴ゛? リンゴ゛
エイゴ゛（オワリ=99)? beetle
      ニホンゴ゛? カブ゛トムシ
エイゴ゛（オワリ=99)? bicycle
      ニホンゴ゛? シ゛テンシャ
エイゴ゛（オワリ=99)? ■
```

実行させた状態は前ページのとおりです。ずいぶん単語を登録できるようになりましたが、入力したデータが上から下まで表示されて、少々見ばえが悪いようです。もうひと工夫して、**単語を５つ入力したら画面をクリアする**ようにしましょう。ＬＩＳＴコマンドでプログラムを画面に表示してみましょう。

```
10 ' file name ==> EITAN1
20 ' 1984.6.10
30 ' sequential data write
40 ' save "0:EITAN1"
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
115 OPEN "O",#1,"1:EITANDAT"
120 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
125 IF A$="99" THEN GOTO 150
130 INPUT "         ﾆﾎﾝｺﾞ";B$
140 PRINT #1,A$;",";B$
145 GOTO 120
150 CLOSE #1
160 END
```

行番号140と145の間に**カウンタ**を入れて、カウンタが５になったら画面をクリアするようにします。このとき**カウンタを０にもどしておく**のを忘れないようにしてください。

カウンタ

```
142 C=C+1
143 IF C=5 THEN C=0 : CLS
```

これで完成です。最後に行番号100以降がふぞろいですのでリナンバリングして整理しておきましょう。

```
RENUM 100,100
```

プログラムリストを右ページに示します。

めんどうでも前ページのように「１：ＥＩＴＡＮＤＡＴ」をＫＩＬＬしてからプログラムをＲＵＮさせ、次の単語を入力してデータファイルを作ってみてください。

```
airplane・・・・・・・・ﾋｺｰｷ        apple・・・・・・・・・・ﾘﾝｺﾞ
beetle・・・・・・・・・ｶﾌﾞﾄﾑｼ      bicycle・・・・・・・・ｼﾞﾃﾝｼｬ
blackboard・・・・・・ｺｸﾊﾞﾝ       castle・・・・・・・・・ｵｼﾛ
city・・・・・・・・・・ﾄｼ          college・・・・・・・・ﾀﾞｲｶﾞｸ
comic book・・・・・・ﾏﾝｶﾞ        desk・・・・・・・・・・ﾂｸｴ
```

```
10 ' file name ==> EITAN1
20 ' 1984.6.10
30 ' sequential data write
40 ' save "0:EITAN1"
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
120 OPEN "O",#1,"1:EITANDAT"
130 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
140 IF A$="99" THEN GOTO 200
150 INPUT "          ﾆﾎﾝｺﾞ";B$
160 PRINT #1,A$;",";B$
170 C=C+1
180 IF C=5 THEN C=0 : CLS
190 GOTO 130
200 CLOSE #1
210 END
```

PRINT#

上に示したプログラムでは、160行で入力データをディスクに書き込んでいますが、このとき使う**PRINT#文**の使い方で間違いやすいケースがありますので、ディスク上でのデータイメージを説明しながら補足しておきます。

まず、例として4つの書き込み方を示します。

```
❶ A$="desk":B$="ﾂｸｴ":PRINT #1,A$,B$
❷ A$="desk":B$="ﾂｸｴ":PRINT #1,A$;B$
❸ A$="desk":B$="ﾂｸｴ":PRINT #1,A$;",";B$
❹ A=291:B=1257:PRINT #1,A;B
```

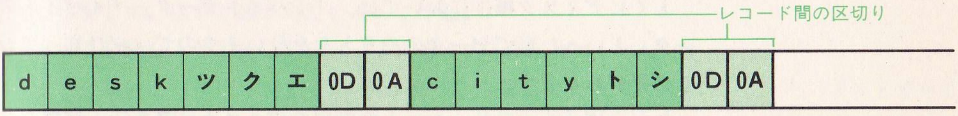
文字列の場合（❶〜❸）

数値の場合（❹）

● 0D、0AはそれぞれCR（キャリッジリターン）、LF（ラインフィード）の16進コードです。

ディスクに書き込まれたシーケンシャルデータを2・6節で説明する**INPUT#文**（インプットシャープ、またはインプットナンバー）で読み出すとき、各**レコード**（この場合、A$とB$で1レコード、AとBで1レコードと考えてください）はレコード間に「0D」「0A」という文字コードがあるかどうかによって、また各**項目**（A$、B$、A、Bなど）はコンマあるいはスペースがあるかどうかによって識別されます。

❷の場合、**A＄、B＄という２つの項目の間にセミコロンがあります**から、ディスクに書き込まれる場合A＄のうしろにすぐB＄が書き込まれますので、この例では「desk ツクエ」で１項目として登録されてしまいます。続いてそのうしろに0D、0Aという16進コードが入り、次のレコードとの区切りになります。



ですから上図の場合、ディスクから読み出すときA＄に「desk ツクエ」が入り、B＄に「city トシ」が入ってしまい、２つの項目deskとツクエを別々に読むことができなくなってしまいます。

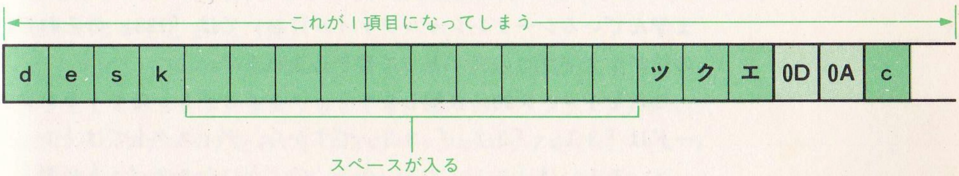
❸の例では**ダブルクォーテーションでかこまれたコンマがあります**から、このコンマがdeskとツクエの間に書き込まれるため正常にディスクから読むことができます。１レコード内に連続して文字列の項目がある場合には必ずこのように区切ってください。

A＄、B＄という２つの文字列を次のようにそれぞれマルチステートメントで書き込めば、A＄、B＄はそれぞれ別々のレコードになりますから問題ありません。

```
PRINT #1,A$ : PRINT #1,B$
```

いちばん間違いやすいのが❶です。

A＄とB＄の間にコンマが入っていますので、この２つの項目が別々に書き込まれそうですが、deskとツクエの間にスペースが入り、このスペースも同一のレコードとして書き込まれてしまいます。



❹の数値データの場合には、**数値間に自動的にスペースが入る**ようになっていますので問題ありません。

それではこのプログラムをセーブして次へ進みましょう。

```
SAVE "EITAN1" ⏎
```

2・5 「バイト」という単位について

バイト

みなさんはもうごぞんじかもしれませんが、コンピュータでは「バイト」という単位をよく使いますので簡単に説明しておきます。

とくにディスク操作においては、「1バイトデータ」「4バイトデータ」といった形でデータのやりとりを行いますので、ぜひ知っておく必要があります。

たとえば「DISK」という文字列をディスクに書き込んだ場合、この項目は「4バイトデータ」ということになります。つまり、文字（ストリング）データの場合、1文字が1バイトになります。

マニュアルの「キャラクタコード表」をごらんください。「D」のキャラクタコードは16進表記で「44」となっていますね。これは16進数になおした場合の「1バイトデータ」です。このように2桁以下の16進数はすべて1バイトで、もっとも大きいのが「FF」です。

ではこれを10進数に換算してみましょう。16進数の「F」は10進数では「15」ですから

&HFF＝15×16＋15＝255

となります。つまり、10進数では255までの整数を1バイトといいます。

話をもとにもどしますが、文字列「DISK」をファイルに書き込むと、イメージ的には「D　I　S　K」あるいは「44　49　53　4B」のように記録され4バイトのメモリを必要とします。

一方、数値データの場合、あとで述べるランダムファイルにおいては、たとえば10進数の「123」は16進数になおすと「7B」ですから、ディスク上に「7B」と記録され1バイト専有しますが、私たちがいま学んでいるシーケンシャルファイルにおいては「123」のそれぞれの位が1文字ずつの文字として書き込まれます。

これをイメージ的に表現しますと、「1」「2」「3」のキャラクタコードは「31」「32」「33」ですから、ディスク上では「1　2　3」あるいは「31　32　33」のようになり3バイト必要とします。

以上、「1バイト」という考え方について簡単に説明しましたが、わからない人はそのままでけっこうです。「1文字が1バイトなんだなあ」くらいに考えておいてください。

2・6　ディスクのデータを読む

この節では、2・4節でディスクに書いたデータを読んで、それをディスプレイに表示してみます。実際に書き込みに成功したかどうかまだ不安です。はやく確かめたいものですね。

データを読むときも先ほどのノートの考え方と同じです。つまり、ノートを開いて（ＯＰＥＮして）、読み終ったら閉じる（ＣＬＯＳＥする）のです。ただしこの場合、モードが**読み出しのモード（インプットモード）**になります。形は次のとおりです。

インプットモード

```
OPEN "I",#1,"1:EITANDAT"
```

（"I" ← インプットモード）

このステートメントに＃１というファイル番号がついていますが、これについて補足説明をしておきます。

コンピュータがディスクの読み書きをする場合、実は直接行うのではなく本体内の**バッファエリア**という場所を中継して行うのです。このバッファエリアの数はディスク起動時の

バッファエリア

```
How many disk files(0-15)?
```

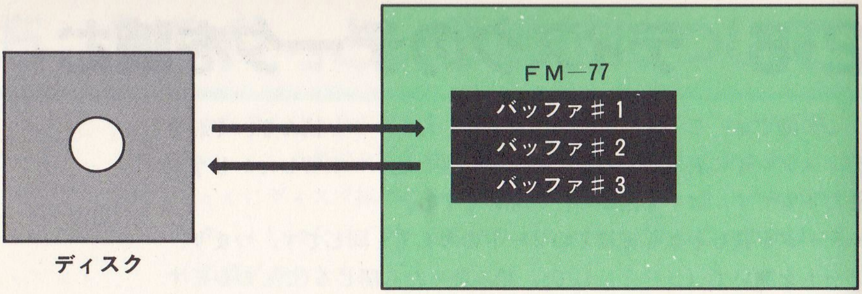
で指定した数だけ確保されます。

上のメッセージでは（０－15）となっていますから16のバッファエリアを使うことが可能ですが、実際のプログラムでファイルを同時にこんなにたくさんＯＰＥＮすることもありませんし、また１つのバッファで256バイト必要ですから、メモリもそれだけよけいにとられてしまいます。

普通は３くらいを入力しておいて、この範囲内でうまくやりくりするほうがよいでしょう。**（注）**

たとえば、１つのプログラムで６つのファイル（仮にＡ、Ｂ、Ｃ、Ｄ、Ｅ、Ｆとしておきます）を使うとしても、＃１のバッファエリアをＡに、＃２のバッファエリアをＢに、＃３のバッファエリアをＣに、というように限定して割り当てられるのではなく、たとえば、＃２のバッファエリアを使っているファイルＢをＣＬＯＳＥすれば、ここを残りのＡ、Ｃ、Ｄ、Ｅ、Ｆは自由にＯＰＥＮして使うことができるのです。つまり、３つのバッファエリアを同時に４つ以上のファイルの入出力で使いさえしなければ、どう使おうと自由なのです。

プログラムの流れの中で、使っていないファイルはＣＬＯＳＥして、バッファエリアをほかのファイルに貸してあげればよいでしょう。



また、どんなにデータ量がぼう大であっても、１つのバッファには256バイトの広さしかありませんから、これを単位として読み書きを行っています。

話をもとに戻しましょう。

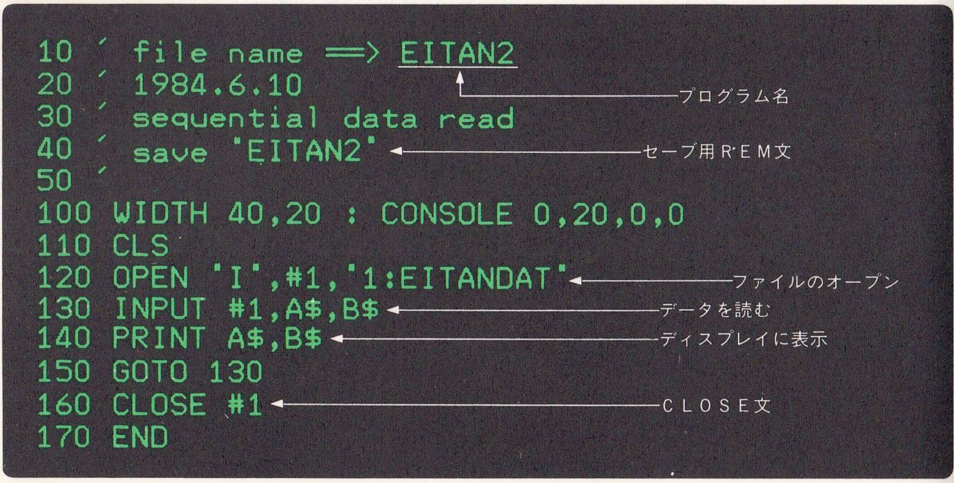
INPUT＃

インプットモードでファイルをＯＰＥＮしましたら、次はデータを読まなければなりませんが、書式は次のようになります。

```
INPUT #1,A$,B$
```

ＩＮＰＵＴ＃文の場合は、うしろに続く変数は数値変数であれ文字変数であれ、コンマで区切ればよいのです。

このプログラム名を「ＥＩＴＡＮ２」として、ディスクからデータを読むプログラムを作りますと次のようになります。



前節で書き込んだデータを読んでディスプレイに表示するようにしてあります。47ページのデータは書き込んでありますか？

それではRUNしてみてください。

おやおや、またピーッというブザー音とともに「Ready」の表示がでてコマンド待ちとなりました。

```
airplane        ヒコーキ
apple           リンゴ
beetle          カブトムシツ
bicycle         ジテンシャ
blackboard      コクバン
castle          オシロ
city            トシ
college         ダイガク
comic book      マンガ
desk            ツクエ

Input Past End In 130
Ready
█
```

ファイルに書かれていたデータ

エラーメッセージ

このエラーメッセージ、

```
Input Past End In 130
```

は、ディスクを扱うようになると、初めの頃よく出会うメッセージです。意味は**「行番号130で(ファイルの)終りを過ぎているのにまだ(データを)読もうとしている(INPUTしようとしている)」**といったところです。

プログラムを見てください。

130行でデータを読んで140行で画面に表示し、150行で再び読み込み処理へ飛ばしています。これではファイルにデータがなくなってもまだ読み込もうとしてしまうはずです。どこかで**ファイルの末尾をチェック**させて、このループから抜け出さなければなりません。

ＥＯＦ関数

この、データチェックのための関数が**ＥＯＦ**(End of fileの略)です。コンピュータは、この関数の値が－1（真、つまりファイルの末尾であるということ）か0（偽、つまりまだファイルがあるということ）かによって判断しているのです。

ですから、この関数をデータを読むステートメントの前に入れなければなりません。

```
125 IF EOF(1) THEN GOTO 160
```

ＥＯＦ(１)、つまり＃１のファイルがもしEnd of file ならば、ファイルのＣＬＯＳＥをします。そして、行番号150のＧＯＴＯ文を変更して、125行へ飛ばすようにします。

```
150 GOTO 125
```

プログラムを実行してみましょう。今度はエラーもなく無事に単語を画面に表示できたと思います。

しかしこれでは、ディスクからデータを読んで、それをディスプレイに表示したというだけで、英単語を表示し、それに対して答えるという形式がととのっていません。少し手直ししましょう。

140行では英単語だけを表示し、その意味を声を出して答え、答え終ったら ⏎ キーを押し、続いて正解を表示するようにステートメントの変更と追加をします。

```
140 PRINT A$
145 INPUT "  ﾆﾎﾝｺﾞ ｦ ﾄﾞｳｿﾞ (OK=RET)";R$
147 PRINT "  ｺﾀｴ ==> ";B$
```

さあ、キーインできましたか？

できたらＲＵＮしてみてください。

```
airplane
  ﾆﾎﾝｺﾞ ｦ ﾄﾞｳｿﾞ (OK=RET)?
  ｺﾀｴ ==> ﾋｺｰｷ
apple
  ﾆﾎﾝｺﾞ ｦ ﾄﾞｳｿﾞ (OK=RET)?
  ｺﾀｴ ==> ﾘﾝｺﾞ
beetle
  ﾆﾎﾝｺﾞ ｦ ﾄﾞｳｿﾞ (OK=RET)?
  ｺﾀｴ ==> ｶﾌﾞﾄﾑｼｯ
bicycle
  ﾆﾎﾝｺﾞ ｦ ﾄﾞｳｿﾞ (OK=RET)?█
```

これで、データを読み出すという１つの仕事を処理するプログラムとしては完成なのですが、問題と正解が画面をスクロールするだけではちょっと感じがでません。１問答えたら画面を消去し、次の出題というふうにしてみましょう。

また、表示位置もＬＯＣＡＴＥ文を使って調整してみました。

次に**変更・追加ステートメント**を示します。

```
140 LOCATE 7,5 : PRINT A$
145 LOCATE 10,6 : INPUT "  ﾆﾎﾝｺﾞ ｦ ﾄﾞｳｿﾞ
(OK=RET)";R$
147 LOCATE 10,8 : PRINT "  ｺﾀｴ ==> ";B$
148 FOR I=1 TO 1000 : NEXT I ←――――時間つぶし
149 CLS ←――――画面をクリアして次の問題に
```

行番号148のＦＯＲ～ＮＥＸＴ文は**時間つぶし**の常とう手段です。このステートメントを入れておかないと、すぐ次に画面消去のステートメントがありますので、行番号147のコタエはほとんど表示されないように見えます。

時間つぶし

ちょっと退屈な勉強をしたのですから、労をねぎらう意味で、「ゴクロウサマデシタ」となぐさめの言葉を入れておきましょう。

```
165 LOCATE 15,7 : PRINT "ｺﾞｸﾛｳｻﾏ ﾃﾞｼﾀ"
166 FOR I=1 TO 3000 : NEXT I
```

そしてリナンバリングとセーブをして完了です。

```
RENUM 130,125 ⏎
SAVE "EITAN1" ⏎
```

```
10 ' file name ==> EITAN2
20 ' 1984.6.10
30 ' sequential data read
40 ' save "EITAN2"
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
120 OPEN "I",#1,"1:EITANDAT"
130 IF EOF(1) THEN GOTO 210
140 INPUT #1,A$,B$
150 LOCATE 7,5 : PRINT A$
160 LOCATE 10,6 : INPUT "  ﾆﾎﾝｺﾞ ｦ ﾄﾞｳｿﾞ
(OK=RET)";R$
170 LOCATE 10,8 : PRINT "  ｺﾀｴ ==> ";B$
180 FOR I=1 TO 1000 : NEXT I
190 CLS
200 GOTO 130
210 CLOSE #1
220 LOCATE 15,7 : PRINT "ｺﾞｸﾛｳｻﾏ ﾃﾞｼﾀ"
230 FOR I=1 TO 3000 : NEXT I
240 END
```

2·7 データを追加する

これまでに私たちは、フロッピィディスクにデータを書き込み、それを読み出して画面に表示するところまで学習してきました。

データ量が多くて、とてもひと息に入力できないとか、何かの都合で次回にまわしたいとか、いろいろ事情も生じてくると思います。

残りのデータを現在作成されているデータファイルに後日あらためて追加して書き込みたいという場合どうしたらよいでしょうか。

ＯＵＴＰＵＴモードではデータ書き込みはできましたが、このモードでは初回の書き込みしかできませんでした。

APPEND

データをシーケンシャルファイルに追加して書き込みたい場合は、**APPEND(アペンド)モード**を使います。

```
OPEN "A",#1,"1:EITANDAT"
```

(注: "A" ― データ追加のモード)

すでにディスクの中にある**「EITAN DAT」**というデータファイルを＃１のバッファを使ってＡＰＰＥＮＤモードでＯＰＥＮしなさい、という意味です。

そして、このＯＰＥＮ文が実行されると、データを追加して書き込みできる準備が完了したことになります。

ＡＰＰＥＮＤモードはＯＵＴＰＵＴモードと同じように、データを書き込むという点においては変わりありませんから、その他のステートメントは同じで、ＯＵＴＰＵＴ用のプログラムを**スクリーンエディタ**で多少変更するくらいで使うことができます。

それでは書き込み用のプログラムファイル**「EITAN1」**をロードして、スクリーンエディタで修正してみましょう。

```
LOAD "EITAN1" ↵
```

「LOAD"」は PF6 キーに定義されていますから、これを使うと便利ですよ。

次に PF2 キーを押してプログラムリストを画面に表示しますが、リストが少しスクロールしてしまいますので、画面モードを80桁×25行にしておきます。

```
WIDTH 80,25 ↵
```

では（PF2）キーを押してください。次のようにリストが表示されます。

```
LIST

10 ′ file name ==> EITAN1
20 ′ 1984.6.10
30 ′ sequential data write
40 ′ save "0:EITAN1"
50 ′
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
120 OPEN "O",#1,"1:EITANDAT"
130 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
140 IF A$="99" THEN GOTO 200
150 INPUT "          ﾆﾎﾝｺﾞ";B$
160 PRINT #1,A$;",";B$
170 C=C+1
180 IF C=5 THEN C=0 : CLS
190 GOTO 130
200 CLOSE #1
210 END

Ready
█
```

上のプログラムリストの中で、白マドになっているところだけ修正すればよいのです。プログラム名は「EITAN3」とします。

カーソル移動キーを使って、次の緑色の文字のように変更してください。

```
10 ′ file name ==> EITAN3
30 ′ sequential data append
40 ′ save "EITAN3"
120 OPEN "A",#1,"1:EITANDAT"
```

〈注〉30行のドライブ番号の指定はあってもなくてもよいので削除することにします。

120行の「1」の修正まで終ったら、そこで（↵）キーを押してください。これで修正作業終了です。LISTコマンドでもう一度確認してみてください。

ではこのプログラムを「EITAN3」としてセーブしておきますが、簡単なセーブ方法があります。カーソルを行番号40の「4」に重ねて

```
40 ' save "EITAN3"
```

↑ここにカーソルを重ね (DEL) キーを5回たたいてリターン

どうですか？　これで**セーブ完了**です。

それではさっそくためしてみましょう。次の基本単語を追加することにします。 (カナ) キーの切り換えを正確にしてください。

```
friend・・・・・・・トモダ゛チ      iris・・・・・・・・・・アヤメ
kindergarten・ヨウチエン       lily・・・・・・・・・・ユリ
map・・・・・・・・・・・・チズ゛         pear・・・・・・・・・・ナシ
```

実行結果は、次のような画面になって終りです。

```
エイゴ゛ (オワリ=99)? pear
        ニホンゴ゛? ナシ
エイゴ゛ (オワリ=99)? 99

Ready
■
```

画面にReadyがでています。しかし、これはシーケンシャルファイルへのデータの追加が確実にできたことの保証ではありません。パソコンは処理が終わればいつもReadyのサインをだすのですから……。まだまだ安心できません。前節で作った読み出しのプログラムで確かめてみましょう。

ロードしてRUNする方法には次の２つがあります。前に勉強しましたからわかりますね。どちらかを実行してください。

```
LOAD "EITAN2",R ↵

RUN "EITAN2" ↵
```

最終的には次の画面が表示され、次に「ゴクロウサマ」が表示されて終りです。

```
      pear
           ニホンゴ゛ ヲ ト゛ウソ゛ (OK=RET)?

           コタエ ==> ナシ
```

データ追加のプログラムを実行するとおわかりのように、画面に突然データ入力をうながすメッセージがでてきて、ちょっととまどいを感じる方もいらっしゃるかもしれません。もう少し余裕をもたせてみましょう。要するに、もう少しパソコンと取り組んでいるという感じを持たせたいものです。

1つは、しばし画面に「データ　ノ　ツイカ　ヲ　シマス」というメッセージを入れること。

もう1つは、最終登録単語と登録語数を表示し、それを確認してから追加処理に入ることです。

ここで「EITAN3」をロードしてください。

画面構成は次のようにしたいと思います。

```
        デ゛ータ ノ ツイカ ヲ シマス

  マツビ゛ ノ タンゴ゛ ==> pear
  トウロク ソウスウ   ==> 16 words

        OK ナラ RETキー ヲ ド゛ウソ゛? ■
```

これら2つの処理は、画面消去をしてからアペンドモードでファイルをOPENするまでの間、つまり行番号110と120の間で行わなければなりません。しかし、この処理ルーチンは少し行数もふえそうですし、現在のスペースではたりそうもありません。RENUMコマンドで行番120を140行にしてスペースをとっておきましょう。

```
RENUM 140,120 ⏎
```

このくらいアキがあれば間に合いそうです。

まず、次のステートメントをキーインします。

```
112 LOCATE 10,5 : PRINT "デ゛ータ ノ ツイカ ヲ シマス"
```

次に、データファイルの末尾にある単語と、登録してある単語を変数に代入しなければなりません。

これはインプットモードでファイルをOPENしてデータを読み飛ばし、ディスプレイに表示しないようにすればよいと思います。末尾の単語は当然読み出し用の文字変数に残りますから、これを画面に表示すればよいのです。また、読み出しルーチンでカウンタを入れておけば、登録してある単語の総数も得られますね。

さっそくコーディングしてみましょう。

ディスクのデータを読む部分は次のとおりです。前節で学習ずみですので、すぐ理解できると思います。

```
114 OPEN "I",#1,"1:EITANDAT"
115 IF EOF(1) THEN GOTO 119 ←データがなければCLOSE
116 INPUT #1,A$,B$
117 C=C+1 ←カウンタ
118 GOTO 115
119 CLOSE #1
```

ごらんのように、行番号116でデータを読んでいますが、画面に表示はしていません。A＄、B＄には、前のデータが追い出されて新しいデータが代入されるというように、先頭のデータから末尾のデータへと仕事がくりかえされ、**最終的には一番最後のデータが残る**ことになります。

カウンタ

Cはカウンタで、**英単語と日本語を１組としてカウント**しています。

次に画面表示部分のプログラムを示します。

```
120 PRINT ←改行
121 LOCATE 7,7:PRINT "ﾏﾂﾋﾞ ﾉ ﾀﾝｺﾞ ==> ";A$
122 LOCATE 7,8:PRINT "ﾄｳﾛｸ ｿｳｽｳ  ==>";C;"words"
123 PRINT ←改行
124 LOCATE 12,10:INPUT "OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ";R$
125 CLS ←画面クリア
```

Cは数値変数ですから符号分も出力され、ここにスペースを入れると英単語とアタマゾロエになりません。

これでほぼ完成ですが、もう１つ大切なことがあります。それは、このあとのメイン処理でカウンタCを使っていますので、これを**ゼロクリア**しておかなければいけないことです。

```
126 C=0
```

そして総仕上げ、リナンバリングです。

行番号がだいぶふぞろいですので、きちんとつけなおしておきましょう。

```
RENUM 120,112
```

行番号112を行番120とし、10きざみにリナンバリングするというわけです。**このプログラムをセーブする**のをお忘れなく……。

プログラムリストを次に示します。

```
10 ' file name ==> EITAN3
20 ' 1984.6.10
30 ' sequential data append
40 ' save "EITAN3"
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
120 LOCATE 10,5 : PRINT "ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ"
130 OPEN "I",#1,"1:EITANDAT"
140 IF EOF(1) THEN GOTO 180
150 INPUT #1,A$,B$
160 C=C+1
170 GOTO 140
180 CLOSE #1
190 PRINT
200 LOCATE 7,7:PRINT "ﾏﾂﾋﾞ ﾉ ﾀﾝｺﾞ==> ";A$
210 LOCATE 7,8:PRINT "ﾄｳﾛｸ ｿｳｽｳ  ==>";C;"words"
220 PRINT
230 LOCATE 12,10:INPUT "OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ";R$
240 CLS
250 C=0
260 OPEN "A",#1,"1:EITANDAT"
270 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
280 IF A$="99" THEN GOTO 340
290 INPUT "          ﾆﾎﾝｺﾞ";B$
300 PRINT #1,A$;",";B$
310 C=C+1
320 IF C=5 THEN C=0 : CLS
330 GOTO 270
340 CLOSE #1
350 END
```

次の単語を入力してテストしてみてください。

私たちは、OUTPUTモードからはじめて、ちょうど20語登録したことになります。

```
poster······ﾎﾟｽﾀｰ     seat········ｻﾞｾｷ
student·····ｶﾞｸｾｲ     wolf········ｵｵｶﾐ
```

●ブレークキーをうっかりして押してしまったとか、シンタックスエラーでプログラムが止まってしまったときは、必ず「CLOSE」をダイレクトモードで入力し、リターンキーを押してください。

2·8 データの修正

これでシーケンシャルファイルへのデータ書き込み・追加、そして読み出すこともできるようになりました。

しかし、これでもまだ不備な点があるのです。それは何かと申しますと、OUTPUTモード、APPENDモードでの書き込みのとき、画面では5単語を1組にして表示し、終われば次の画面へというようにしておりましたが、単語は英語・日本語をペアにしてそのつど書き込んでいました。つまり、キーインしたスペルが正しかったかどうかの確認をしていません。

前にもお話ししましたが、シーケンシャルファイルでは一度書き込まれたものはあとで修正するというわけにはまいりません。ですから、なおいっそう慎重を期したいものです。苦労して打ち込んだデータの中にミスがあったら、あまりいい気分はしませんね。

〈注〉修正がまったくできないというわけではありませんが、そのためには配列を使いますのでデータ量が多くなるとメモリが不足してきます。

そこで本節では、修正するかどうかのルーチンを作って確認をうながすようにプログラム「EITAN3」を手直ししてみましょう。

「EITAN1」にも必要な部分がありますが、重複することと、利用度は「EITAN3」のほうが高いので、こちらを選びました。

DELETE

まず、「EITAN3」の行番号250から350までは作りなおさなければなりませんので、削除をします。DELETEコマンドを使えばいいですね。

```
DELETE 250-350 ⏎    ← 行番号250から350までを削除
```

行番号10、20、40を修正してリストをとると次のようになります。

```
10 ' file name ==> EITAN4      ← 新しいファイル名
20 ' 1984.6.10
30 ' sequential data append
40 ' save "EITAN4"             ← 新しいファイル名
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
120 LOCATE 10,5 : PRINT "デ-タ ノ ツイカ ヲ シマス"
130 OPEN "I",#1,"1:EITANDAT"
140 IF EOF(1) THEN GOTO 180
150 INPUT #1,A$,B$
160 C=C+1
```

```
170 GOTO 140
180 CLOSE #1
190 PRINT
200 LOCATE 7,7:PRINT "ﾏﾂﾋﾞ ﾉ ﾀﾝｺﾞ==> ";A$
210 LOCATE 7,8:PRINT "ﾄｳﾛｸ ｿｳｽｳ  ==>";C;"words"
220 PRINT
230 LOCATE 12,10:INPUT "OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ";R$
240 CLS
```

行番号120以降はサブルーチンにします。 **サブルーチン**

まず、次のステートメントをキーインしてください。

```
115 '
116 ' last data read     ←サブルーチンの見出し
117 '
```

サブルーチンの見出しの前後には**REM文**を入れておくと見やすくなります。また、サブルーチンには必ず**RETURN文**をつけなければいけませんでした。

```
250 RETURN
```

これでサブルーチンとしての形ができあがりました。**RENUM**を使って、このサブルーチンを行番号1000からにしましょう。

```
RENUM 1000,115
```

リナンバリングを実行したことによって、行番号1000以降は次のようになりました。途中は省いて前後だけ示してあります。

```
1000 '
1010 ' last data read
1020 '
1030 LOCATE 10,5 : PRINT "ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ"
1040 OPEN "I",#1,"1:EITANDAT"


1140 LOCATE 12,10:INPUT "OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ";R$
1150 CLS
1160 RETURN
```

次にデータ入力の部分を作ります。プログラムは次のようになります。

```
120 GOSUB 1000
130 DIM AA$(10),BB$(10)
140 OPEN "A",#1,"1:EITANDAT"
150 FOR I=1 TO 5
160   INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";AA$(I)
170   IF AA$(I)="99" AND I=1 THEN GOTO 220
180   IF AA$(I)="99" AND I<>1 THEN N=I-1
              : GOSUB 2000 : GOTO 220
190   INPUT "          ﾆﾎﾝｺﾞ";BB$(I)
200 NEXT I : N=I-1
210 GOSUB 2000 : GOTO 150
220 CLOSE : END
```

まず最初に行番号120で、ファイルを読んで末尾のデータを画面に表示するサブルーチンへ飛びます。これは前節のケースとまったく同じです。

行番号130で配列を宣言していますが、これは、**入力した英語と日本語をそれぞれ記憶し、修正ルーチンで画面に表示させる**ために使うものです。このプログラムでは５つの単語を１セットにして考えていますので、10個以下ですから配列宣言の必要はないのですが、あとあとのことを考えますと、習慣づけておいたほうがよいでしょう。

ＦＯＲ〜ＮＥＸＴ

行番号150と200の間で５組ごとの単語入力の**ＦＯＲ〜ＮＥＸＴループ**を作っています。

ここで気をつけなければいけないのは、行番号190でループから抜け出すとき、Ｉをマイナス１してＮに代入していることです。

次のようなダイレクトステートメントを実行してみてください。

```
FOR I=1 TO 10 : NEXT I : ? I
 11

Ready
█
```

ちょっと考えると、この例の場合ＦＯＲ〜ＮＥＸＴループから抜け出たとき、Ｉの値は10と思いがちですが、実際には11が入ってしまいます。Ｉに10が入っていると勘違いをしてＩをＮに代入して、プログラムのどこかで、ＦＯＲ　Ｊ＝１　ＴＯ　Ｎなどとしますと、Ｊは11回ループすることになりますから、へんなことになってしまいます。

ＦＯＲ〜ＮＥＸＴループを抜け出たとき、あるいは終りのマーク「99」が入力されたとき、データ修正のサブルーチンへ飛ばします。

データ修正のサブルーチンは次のとおりです。

```
2000 '
2010 ' data correct subroutine
2020 '
2030 CLS
2040 LOCATE 5,1 : PRINT "ﾆｭｳﾘｮｸ ﾃﾞｰﾀ ｲﾁﾗﾝ"
2050 FOR J=1 TO N
2060   LOCATE 2,2*J : PRINT 2*J-1;"  ";AA$(J)
2070   LOCATE 2,2*J+1:PRINT 2*J;"  ";BB$(J)
2080 NEXT J
2090 LOCATE 2,13
2100 INPUT "ｼｭｳｾｲ ｼﾏｽｶ (y/n)";ANS$
2110 IF ANS$="N" OR ANS$="n" THEN GOTO 2250
2120 LOCATE 2,13
2130 INPUT "ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ (ｵﾜﾘ=99)";NO
2140 IF NO=99 THEN GOTO 2250
2150 K=(NO+1) ¥ 2
2160 IF NO MOD 2=0 THEN GOTO 2210
2170 LOCATE 2,14
2180 PRINT AA$(K);" ==> "; : INPUT AA$
2190 IF AA$<>"" THEN AA$(K)=AA$
2200 LOCATE 2,14 : PRINT SPC(30) : GOTO 2120
2210 LOCATE 2,14
2220 PRINT BB$(K);" ==> "; : INPUT BB$
2230 IF BB$<>"" THEN BB$(K)=BB$
2240 LOCATE 2,14 : PRINT SPC(30) : GOTO 2120
2250 FOR J=1 TO N
2260   PRINT #1,AA$(J);",";BB$(J)
2270 NEXT J
2280 CLS : RETURN
```

このサブルーチンへジャンプしてきますとまず、画面の一番上に、「ニュウリョク　データ　イチラン」という見出しを表示させます。

そして次の行から、私たちがいま入力した英単語と日本語を頭に通し番号をつけて表示します。これは2050行から2080行のループで行っています。

入力データが表示し終りますと、画面の一番下に、修正するかどうかの確認メッセージを表示します。

修正がなければ、2250行へジャンプし、データをディスクに書き込んで、このサブルーチンからＲＥＴＵＲＮです。

データ修正の必要があるときは、「バンゴウ　ヲ　ドウゾ　（オワ

リ＝９９)」というメッセージが現れますから、修正したいデータの頭についている番号を入力してリターンキーを押します。修正を終りたい場合には「９９」を入力してください。「９９」が入力されると、2250行へジャンプし、データの書き込みを実行します。

さて修正の場合ですが、たとえば５つのデータを入力してこのサブルーチンへ飛んできたとすると、英単語、日本語に通し番号がつけられ、次のように画面に表示されます。

1	station	――ＡＡ＄(１)の内容
2	エキ	――ＢＢ＄(１)の内容
3	beach	――ＡＡ＄(２)
4	ハマベ	――ＢＢ＄(２)
5	ship	――ＡＡ＄(３)
6	フネ	――ＢＢ＄(３)
7	tiger	――ＡＡ＄(４)
8	トラ	――ＢＢ＄(４)
9	bird	――ＡＡ＄(５)
10	トリ	――ＢＢ＄(５)

ここで修正したい語の前についている番号を入力しますから、それが変数ＮＯに代入されます(2130行)。しかし、この番号は配列の添字１～５とはうまく対応していません。

そこで、2150行の算式

K= (NO+1) ¥2

により、入力された番号に対応する配列の添字を計算します。

円マーク(¥)

円マーク「¥」は整数除算の商を求めるものでしたね。この計算式によって１～10の整数の商は次のようになります。

整数	1	2	3	4	5	6	7	8	9	10
商(K)	1	1	2	2	3	3	4	4	5	5

これで、うまく配列の添字に対応させることができました。

しかし、これだけではまだ、英単語の修正か日本語の修正かの判断ができません。

MOD

このチェックを2160行で行っています。ＭＯＤは整数除算の剰余を求めるものです。つまり、入力された番号を２で割ってその余りが０なら日本語の修正（配列はＢＢ＄）、１なら英単語の修正（配列はＡ

A＄）ということになります。

2170〜2200行が英単語の修正、2210〜2240行が日本語の修正です。

修正方法は、間違えた語がまず表示されますから、それを見ながら正しいデータを入力するようになっています。

1語の修正が終るとそれをＳＰＣ(スペース)関数で消去して、ふたたび番号入力に戻ります。

ＳＰＣ関数

修正作業が終ったら、ディスクにデータを書き込んで、このサブルーチンから戻ります。

プログラムの実行

それではプログラムをセーブしてから(PF 3)キーで走らせてみましょう。最初に次の初期画面となります。

```
        ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ

      ﾏﾂﾋﾞ ﾉ ﾀﾝｺﾞ ==> wolf
      ﾄｳﾛｸ ｿｳｽｳ   ==> 20 words

          OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ? ■
```

⏎キーを押すと単語入力モードになりますから、次のように5つの単語を入力してみましょう。beach を間違えて beech と入力してしまいました。これは修正ルーチンでなおすことにします。

```
ｴｲｺﾞ (ｵﾜﾘ=99)? station
        ﾆﾎﾝｺﾞ? ｴｷ
ｴｲｺﾞ (ｵﾜﾘ=99)? beech
        ﾆﾎﾝｺﾞ? ﾊﾏﾍﾞ
ｴｲｺﾞ (ｵﾜﾘ=99)? ship
        ﾆﾎﾝｺﾞ? ﾌﾈ
ｴｲｺﾞ (ｵﾜﾘ=99)? tiger
        ﾆﾎﾝｺﾞ? ﾄﾗ
ｴｲｺﾞ (ｵﾜﾘ=99)? bird
        ﾆﾎﾝｺﾞ? ■
```

aをeと入力してしまった

birdに対する日本語「トリ」を入力し ⏎ キーを押すと画面は次のようになります。

```
  ﾆｭｳﾘｮｸ ﾃﾞｰﾀ ｲﾁﾗﾝ
 1   station
 2   ｴｷ
 3   beech
 4   ﾊﾏﾍﾞ
 5   ship
 6   ﾌﾈ
 7   tiger
 8   ﾄﾗ
 9   bird
 10   ﾄﾘ

ｼｭｳｾｲ ｼﾏｽｶ (y/n)? ■
```

beeehを修正しなければなりませんから当然「y」を入力します。

すると画面の下のほうに番号入力のメッセージが表示されます。

```
ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ (ｵﾜﾘ=99)? ■
```

ここで「3」をキーインしリターンキーを押しますと、次のように間違えて入力した単語が表示されますから、カーソル位置から正しいデータを入力してリターンキーを押してください。

```
ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ (ｵﾜﾘ=99)? 3
beech ==> ? ■
```

修正データを入力するとふたたび番号入力になりますが、これで修正作業を終りたければ「９９」をキーインします。するとディスクへのデータ書き込みがあり、画面はクリアされて英単語の入力になります。私たちはこれで終りにしますので「９９」を入力しましょう。

さあ、うまくいったでしょうか？　データ読み出しのプログラムをＲＵＮさせて確認してみてください。

```
RUN "EITAN2" ⏎
```

2·9 メニュープログラムを作ろう

以上、「英単語練習プログラム」を例にとってシーケンシャルファイルについて説明してきました。

アウトプットモード、インプットモード、アペンドモード、いかがでしたか？　思ったほどむずかしくはなかったと思います。

これまでに私たちがドライブ0のディスクにセーブしたプログラムファイルは次の4つです。

```
EITAN1
EITAN2
EITAN3
EITAN4
```

そして、ドライブ2のディスクに書き込んだデータファイルは

```
EITANDAT
```

という、英単語と日本語で構成されたものでした。

さて、今後あなたがこれらのプログラムを使おうとする場合、いまのままですと、どのプログラムがどんな目的で使われるかを覚えておいて、次のような操作によらなければなりません。

```
RUN "EITAN1"

RUN "EITAN2"

RUN "EITAN3"

RUN "EITAN4"
```

ファイル名を打ち間違えて

```
File Not Found
```

というエラーメッセージが返ってきてギクッとしたり、いや、それよりとにかくキーインもめんどうです。もう少し楽をすることはできないでしょうか？

人間は１つのことをマスターすると、そのことをもう少し手をかけないでできないものかとムシのよい考え方をします。しかし、これはおおいにけっこうです。こういった欲がでてきてはじめて前進していくのですから……。

私たちもおおいに思いをめぐらせましょう。

メニュープログラム

そのためには、**「英単語練習プログラム」を管理するメニュープログラム**を作ってしまうことです。

メニュープログラムとは、画面にいろいろな仕事内容を表示させ、その中からどれか１つを選び、希望する仕事を処理していくものです。

たとえば、メニュー画面を次のように構成します。

```
エイタンコ゛ メニュー フ゜ロク゛ラム

タンコ゛ ノ カキコミ・・・・・・・・・・1
タンコ゛ ノ ヨミタ゛シ・・・・・・・・・2
タンコ゛ ノ ツイカ・・・・・・・・・・・3
オワリ・・・・・・・・・・・・・・・・・・4

input job number ? ■
```

このようにしておきますと、選択した番号に応じて、

```
1 ==> RUN "EITAN1"
```

```
2 ==> RUN "EITAN2"
```

```
3 ==> RUN "EITAN4"
```

というように、それぞれのプログラムが自動的に実行されるわけです。

それではさっそくコーディングしてみましょう。

まず出だしのＲＥＭ文から。

```
10 ' file name ==> MENU01
20 ' 1984.6.20
30 ' menu program for ENGLISH words
40 ' save "MENU01"
50 '
```

次に初期設定をしておきます。

```
100 WIDTH 40,25 : CONSOLE 0,25,0,0
110 DEFINT A-Z ←──────── 整数宣言
120 CLS
130 '
```

行番号100のＷＩＤＴＨ文で画面のスクロール行数を25にしてありますが、これはメニュー画面の見出しをケイ囲みにするためにこのようにしました。と申しますのは、スクロール行数が20では行間があいてしまってケイがつながらないからです。

DEFINT

次に行番号110のＤＥＦＩＮＴ文ですが、これは変数の型宣言です。普通このように型の宣言がされていませんと、変数はすべて単精度型とみなされ、処理時間に大きな差がでてきます。数値を扱う場合も整数が主体でしたから、Ａ～Ｚで始まる変数を整数宣言しておいて、単精度、倍精度扱いの変数にだけ「！」「＃」をつけておけば能率的です。あるいは、ＤＥＦＳＮＧ、ＤＥＦＤＢＬなどを使って、あらたに宣言しなおしてもよいでしょう。

DEFSNG
DEFDBL

このプログラムではあまり必要ありませんが、処理時間の短縮という意味で、習慣づけておいたほうがよいものの１つです。

メニューはDATA文で

DATA文

画面に表示するメニューはＤＡＴＡ文にします。

```
1000 '
1010 '  data area
1020 '
1030 DATA 4
1040 DATA ﾀﾝｺﾞ ﾉ ｶｷｺﾐ··········1
1050 DATA ﾀﾝｺﾞ ﾉ ﾖﾐﾀﾞｼ·········2
1060 DATA ﾀﾝｺﾞ ﾉ ﾂｲｶ···········3
1070 DATA ｵﾜﾘ··················4
```

行番号1030の数値４は、メニューの数を表しています。そして、この値をＲＥＡＤ文で最初に読んで変数に入れておけば非常に便利です。このことはあとで説明します。

行番号1040からはデータです。１行１行にしないで、各データをコンマで区切って、１行に３つでも４つでもつなげて書いてもかまいません。ただ、本づくりの体裁上、このようにしました。

文字列のデータの前後に意味のあるスペースがあるとか、データ中

にコンマ、ピリオドなどが含まれる場合は、ダブルクォーテーションでそのデータをくくってください。ここで扱っているデータの中の点は、ピリオドではなくナカグロですから、このままでさしつかえありません。

次に、データを読み込む部分です。

```
140 READ MENU
150 DIM M$(MENU)      ←配列の宣言
160 FOR I=1 TO MENU
170   READ M$(I)      ←配列にメニューを読み込む
180 NEXT I
190 '
```

REDA文

140行で変数MENUにREAD文でメニューの数を読み込みます。

次に、それぞれのメニューを配列にとりこみますので、配列の宣言をしなければなりません。行番号150で配列を宣言してから、READ文でメニューを読みます。

もうおわかりいただけたと思いますが、DATA文の先頭にデータ数を入れておくと、このように、配列宣言、FOR～NEXT文のループ回数などが、変数を使って自動的に設定できてしまうのです。

DIM文

もしデータ数を変数に代入しておきませんと、プログラムの初期設定部分でDIM文を入れておかなければなりませんし、FOR文にも数値を入れなければなりません。たとえば次のように……。

```
115 DIM M$(20)          <注>これはキーインしないでください。
116 '
160 FOR I=1 TO 4
170   READ M$(I)
180 NEXT I
```

メニュー数を使ったFOR～NEXT文は他の部分でもでてくると思いますし、メニューを追加するときなどに、いちいちこの数値を修正しておかなければなりません。短いプログラムでしたら、手間はかかりませんが、長くなるとけっこうめんどうなものです。

また、最初の説明のように、変数を使って配列宣言したほうが、メモリを有効に使えるのです。

これはディスク操作とは関係ありませんが、プログラミングの定石として覚えておきましょう。

メイン部分のプログラム

メニューを画面に表示して、仕事番号を選択させ、希望のプログラムを実行する部分です。プログラムリストは次のとおりです。

```
200 LOCATE 7,4 : PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,5:PRINT "| ｴｲﾀﾝｺﾞ ﾒﾆｭｰ ﾌﾟﾛｸﾞﾗﾑ |"
230 LOCATE 7,6 : PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270   LOCATE 8,7+I : PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13 : INPUT "input job number ";NO
310 LOCATE 8,15 : PRINT M$(NO)
320 LOCATE 18,16 : INPUT "OK (y/n)";Y$
330 IF Y$="N" OR Y$="n" THEN CLS : GOTO 200
340 ON NO GOTO 360,370,380,390
350 '
360 RUN "EITAN1"
370 RUN "EITAN2"
380 RUN "EITAN4"
390 CLS
400 LOCATE 15,7 : PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
410 END
```

見出しの表示（200～240）

メニューを画面に表示（260～280）

仕事番号に対応した行番号へ飛ぶ（340）

プログラムをRUN（360～380）

行番号200から240までは、見出しの表示です。見出しをケイで巻くと体裁がととのいます。

LOCATE文
STRING$関数

ここでは**LOCATE文でカーソル位置を指定し、PRINT文で画面に表示**しています。**STRING$関数**の書式は次のとおりです。

```
STRING$(21,"─")
STRING$(21,149)
```

"─" と 149：文字または数値

21：出力する個数

この例は2つとも同じ結果を出力します。

文字指定の場合には、ダブルクォーテーションでくくってください。また**字数は1字しか有効ではありません**から、3文字入れても最初の1字だけが出力されます。

●「─」はマイナス記号ではありません。グラフィックキャラクタです。

数値指定の場合には、その値が0～255の範囲に限られ、これをキャラクタコードとみなして対応する文字を出力します。この例で「─」の

キャラクタコードは149です。

行番号260から280でメニューを画面に表示します。

ここでもMENUという変数をFOR～NEXT文に使っていることに注意してください。

また、LOCATE文の位置指定にループ用変数Ｉを使っています。これもかなり便利な方法です。

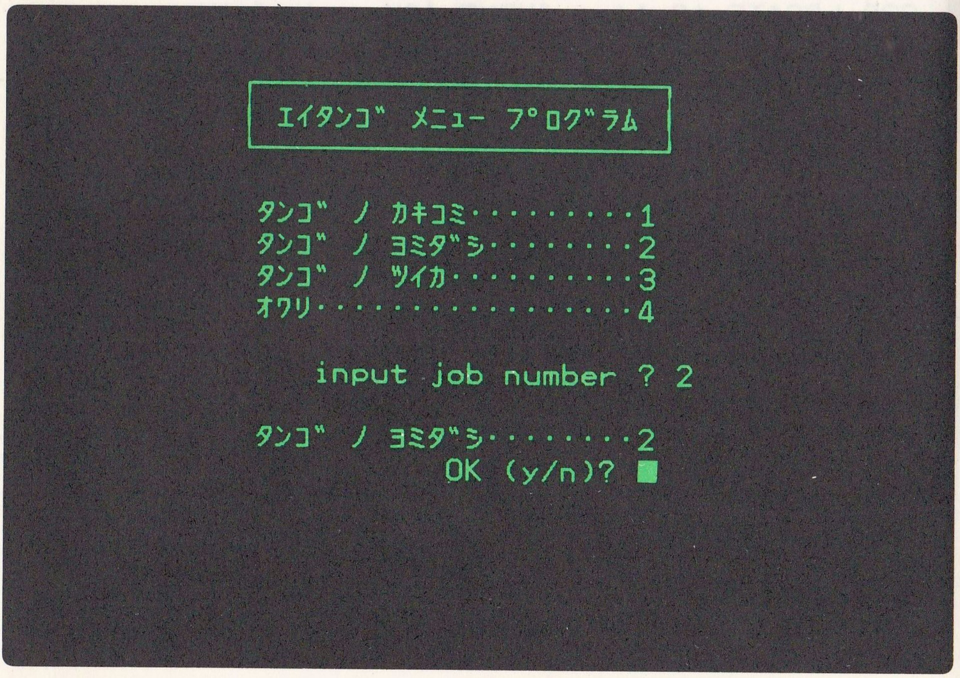
行番号300でメニュー選択をうながし、行番号320では選択したメニューでOKかどうか確認させます。「ノー」であればもう一度200行へ飛ばし、やりなおしです。

ON～GOTO

OKでしたら、340行のON～GOTO文で、メニュー番号に対応した処理へ飛びます。

行番号360から380までが、前に説明したプログラム実行のステートメントです。

さあ、それではこのメニュープログラムをセーブしてからRUNさせてみましょう。



前ページの表示画面は、仕事番号２を選択したので、その確認をうながしているところです。もしよかったら、小文字のｙを入力してみてください。なつかしい「英単語読み出しのプログラム」が走り出します。どうでしたか？　うまくいきましたか？

キーインにミスがなければ必ず動くはずです。

ここで注意を1つ。
メニュー選択の際、1番を選びますと、これはデータ書き込み用プログラムをRUNさせることになりますが、46ページで説明したように、すでに初回の書き込みがすんでいますと、

File Already Exists

のエラーになりますから注意してください。
ここでもう少し欲を出してプログラミングのテクニックを学んでおきましょう。

引き続き他のプログラムを実行したい

私たちがこの節で組んできたプログラムは、たとえば「単語追加」のメニューを選択しますと、追加処理が終るとプログラムはENDになり、追加したデータを読み出したいときは、もう一度メニュープログラムをロードしてRUNさせなければなりません。
何かよい方法はないでしょうか？
あります、あります。**それぞれのプログラムにメニューへ戻るステートメントを折り込んでやればよい**のです。
「書き込み」「読み込み」「追加」の3つのプログラムのENDがある行番号は次のとおりでした。

```
EITAN1 ==>  210 END

EITAN2 ==>  240 END

EITAN4 ==>  220 CLOSE : END
```

ですから、これらのEND文を削除して(CLOSE文は残しておく)、このうしろに**メニューへ戻るステートメントを書く**ことにします。プログラムを次ページに示します。
NEWコマンドでメモリをクリアしてからキーインしていただきますが、その前に**メニュープログラムをセーブしたかどうかを確認**してください。
しつこいようですが、最初のうちは、せっかく入力した新しいプログラムをセーブしないうちに他のプログラムをロードしてしまったとか、NEWを実行しないで他のプログラムを入力してしまったとかの

ミスをおかし後悔することがよくあります。

```
300 '
310 CLS
320 LOCATE 10,6 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 15,8 : PRINT "( y/n ) ?"
340 Y$=INKEY$
350 IF Y$="" THEN GOTO 340
360 IF Y$="Y" OR Y$="y" THEN RUN "MENU01"
370 END
```

簡単なプログラムですね？

行番号320、330でメッセージを表示しています。

行番330のＰＲＩＮＴ文のところはＩＮＰＵＴ文を使ってもいいのですが、新しい知識を吸収するという意味で、あえてＰＲＩＮＴ文にし、340行で**ＩＮＫＥＹ＄関数**を使用しました。

ＩＮＫＥＹ＄関数

この関数は何かキー入力があるまで、行番号340と350の間をループしています。どのキーでもよいですから、キーが押されると行番号360へ落ちていきます。

さあ、それでは**このプログラム名を「ＭＲＧ」としてアスキー形式でセーブし、「ＥＩＴＡＮ１」とつなぎ合わせてみましょう。**

次の手順で行ってください。

```
SAVE "MRG",A      ←――― アスキーセーブする

Ready
LOAD "EITAN1"     ←――― プログラムをロードする

Ready
210               ←――― 210行を削除
MERGE "MRG"       ←――― 「ＭＲＧ」をマージしてつなげる

Ready
SAVE "EITAN1"     ←――― 完成したプログラムをセーブ

Ready
■
```

まったく同じ方法で「EITAN2」「EITAN4」についてもマージを行ってください。無事終って「MRG」が不要になりましたら、削除しておきましょう。KILL文を使うのでしたね。

```
KILL "MRG" ⏎
```

(PF2) キーでリストを表示させてみてください。

```
10  ' file name ==> EITAN1
20  ' 1984.6.10
30  ' sequential data write
40  ' save "0:EITAN1"
50  '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
120 OPEN "O",#1,"1:EITANDAT"
130 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
140 IF A$="99" THEN GOTO 200
150 INPUT "          ﾆﾎﾝｺﾞ";B$
160 PRINT #1,A$;",";B$
170 C=C+1
180 IF C=5 THEN C=0 : CLS
190 GOTO 130
200 CLOSE #1
300 '
310 CLS
320 LOCATE 10,6 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 15,8 : PRINT "( y/n ) ?"
340 Y$=INKEY$
350 IF Y$="" THEN GOTO 340
360 IF Y$="Y" OR Y$="y" THEN RUN "MENU01"
370 END
```

行番号300以降がうまくマージされたようです。

これでこの章は終りです。次章では電話帳プログラムを例にとって、もう少しシーケンシャルファイルに慣れるようにしましょう。

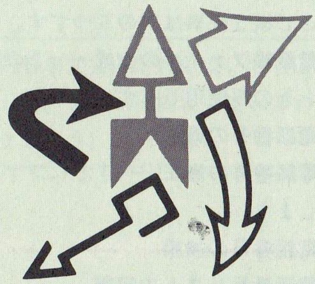
なお、次ページに「英単語練習メニュープログラム」の総リストをのせておきます。

英単語練習メニュープログラム

```
10 ' file name ==> MENU01
20 ' 1984.6.20
30 ' menu program for ENGLISH words
40 ' save "MENU01"
50 '
100 WIDTH 40,25 : CONSOLE 0,25,0,0
110 DEFINT A-Z
120 CLS
130 '
140 READ MENU
150 DIM M$(MENU)
160 FOR I=1 TO MENU
170   READ M$(I)
180 NEXT I
190 '
200 LOCATE 7,4 : PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,5:PRINT "| ｴｲﾀﾝｺﾞ ﾒﾆｭｰ ﾌﾟﾛｸﾞﾗﾑ |"
230 LOCATE 7,6 : PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270   LOCATE 8,7+I : PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13 : INPUT "input job number ";NO
310 LOCATE 8,15 : PRINT M$(NO)
320 LOCATE 18,16 : INPUT "OK (y/n)";Y$
330 IF Y$="N" OR Y$="n" THEN CLS : GOTO 200
340 ON NO GOTO 360,370,380,390
350 '
360 RUN "EITAN1"
370 RUN "EITAN2"
380 RUN "EITAN4"
390 CLS
400 LOCATE 15,7 : PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
410 END
1000 '
1010 ' data area
1020 '
1030 DATA 4
1040 DATA ﾀﾝｺﾞ ﾉ ｶｷｺﾐ·········1
1050 DATA ﾀﾝｺﾞ ﾉ ﾖﾐﾀﾞｼ········2
1060 DATA ﾀﾝｺﾞ ﾉ ﾂｲｶ··········3
1070 DATA ｵﾜﾘ················4
```

第3章

シーケンシャルファイル[II]



3·1 メニュープログラム

前章では「英単語練習プログラム」を例にとってシーケンシャルファイルの扱い方について説明してきました。

この章では、電話帳プログラムを例にとって、シーケンシャルファイルについて、もう少し勉強しておきましょう。

電話帳プログラムとは、名前を入力するとその人の電話番号を表示してくれるものです。考え方は英単語の場合とまったく同じですが、みなさんにもう少し慣れていただくためと、検索テクニックがありますのでとりあげることにしました。また、プリンタも使ってみることにしましょう。

ここで扱う仕事は次の5つです。

❶ **電話帳ファイルの作成**——このプログラムは新規のファイル作成のときのみ使用します。

❷ **電話番号の追加**

❸ **電話番号の修正**——すでにファイルに書き込んであるデータを修正します。

❹ **電話番号の検索**

❺ **電話番号リストの印刷**

メニュー画面は次のようにレイアウトしました。

```
        ﾃﾞﾝﾜﾁｮｳ ﾌﾟﾛｸﾞﾗﾑ

TEL FILE ﾉ ｻｸｾｲ･････1
TEL No ﾉ ﾂｲｶ････････2
TEL No ﾉ ｼｭｳｾｲ······3
TEL No ﾉ ｹﾝｻｸ·······4
TEL LIST ﾉ ｲﾝｻﾂ·····5
ｵﾜﾘ·················6

    input job number ? 1

TEL FILE ﾉ ｻｸｾｲ･････1
           OK (y/n)? ■
```

メニュープログラムは、前章のメニュープログラムの部分的な手直しとDATA文の追加によって作成することにします。LOADコマンドで「MEMU01」をロードしてください。

まず最初の部分は、ファイル名を変更して次のようにします。

```
10 ' file name ==> MENU02
20 ' 1984.6.20
30 ' menu program for phone number
40 ' save "MENU02"
50 '
100 WIDTH 40,25 : CONSOLE 0,25,0,0
110 DEFINT A-Z
120 CLS
130 '
```

新しいファイル名

ここはファイル名とREM文の内容を変えるだけでよいでしょう。

次にメニュー項目が2つふえました。メニュー内容も変わるわけですから、DATA文を作っておきましょう。

```
1000 '
1010 ' data area
1020 '
1030 DATA 6
1040 DATA TEL FILE ノ サクセイ・・・・・1
1050 DATA TEL No ノ ツイカ・・・・・・・・2
1060 DATA TEL No ノ シュウセイ・・・・・・3
1070 DATA TEL No ノ ケンサク・・・・・・・4
1080 DATA TEL LIST ノ インサツ・・・・・5
1090 DATA オワリ・・・・・・・・・・・・・・・・・・6
```

データ数

データ数増加にともなって、行番号340行以降にステートメントの追加・変更をします。

```
340 ON NO GOTO 360,370,380,381,382,390
350 '
360 RUN "TELNO1"
370 RUN "TELNO2"
380 RUN "TELNO3"
381 RUN "TELNO4"
382 RUN "TELNO5"
390 CLS
400 LOCATE 15,7 : PRINT "シゴト ヲ オワリマス"
410 END
```

追加

新しいプログラムファイル名

2つふえたメニューに対するプログラムの実行ステートメントは、行番号380と390の間に入れます。そして**RENUMコマンドで行番号を整理**しておきましょう。

```
RENUM 390,381 ⏎
```

```
RENUM 1000,440 ⏎
```

あとは、**見出しの変更と、LOCATE文を修正して表示位置をなおしておきます。**メニューが2つふえましたから、行番号200から280までのLOCATE文の縦座標位置をマイナス2して、2行分上へ表示することにしましょう。カーソル移動キーを使って5箇所を修正し、最後に ⏎ キーを押します。

```
200 LOCATE 7,2 : PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,3:PRINT "│  ﾃﾞﾝﾜﾁｮｳ ﾌﾟﾛｸﾞﾗﾑ     │"
230 LOCATE 7,4 : PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270   LOCATE 8,5+I : PRINT M$(I)
280 NEXT I
290 '
```

見出しの修正

これでメニュープログラムは完成です。プログラムリストを次に示します。

```
10 ' file name ==> MENU02
20 ' 1984.6.20
30 ' menu program for phone number
40 ' save "MENU02"
50 '
100 WIDTH 40,25 : CONSOLE 0,25,0,0
110 DEFINT A-Z
120 CLS
130 '
140 READ MENU
150 DIM M$(MENU)
160 FOR I=1 TO MENU
170   READ M$(I)
180 NEXT I
190 '
```

```
200 LOCATE 7,2 : PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,3:PRINT "|  ﾃﾞﾝﾜﾁｮｳ ﾌﾟﾛｸﾞﾗﾑ     |"
230 LOCATE 7,4 : PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270   LOCATE 8,5+I : PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13 : INPUT "input job number ";NO
310 LOCATE 8,15 : PRINT M$(NO)
320 LOCATE 18,16 : INPUT "OK (y/n)";Y$
330 IF Y$="N" OR Y$="n" THEN CLS : GOTO 200
340 ON NO GOTO 360,370,380,390,400,410
350 '
360 RUN "TELNO1"
370 RUN "TELNO2"
380 RUN "TELNO3"
390 RUN "TELNO4"
400 RUN "TELNO5"
410 CLS
420 LOCATE 15,7 : PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
430 END
1000 '
1010 ' data area
1020 '
1030 DATA 6
1040 DATA TEL FILE ﾉ ｻｸｾｲ·····1
1050 DATA TEL No ﾉ ﾂｲｶ········2
1060 DATA TEL No ﾉ ｼｭｳｾｲ······3
1070 DATA TEL No ﾉ ｹﾝｻｸ·······4
1080 DATA TEL LIST ﾉ ｲﾝｻﾂ·····5
1090 DATA ｵﾜﾘ··················6
```

実行してみたい方は、360行から400行までのステートメントを次のようにREM文にしてRUNさせてみてください。

アポストロフィを入れてREM文にします

```
360 'RUN "TELENO1"
```

正常に動いたらアポストロフィをとってセーブしておきましょう。

```
SAVE "MENU02"
```

3・2　電話帳ファイルの作成

電話帳ファイルの作成プログラムは、データを次々に入力し、それをディスクに書き込めばよいわけですから、77ページのプログラムにちょっと手を加えればできてしまいます。

OUTPUTモードでシーケンシャルファイルをOPENし、INPUT文のメッセージを変えればよいでしょう。

次のようなプログラムになります。

```
10 ' file name ==> TELNO1
20 ' 1984.6.10
30 ' sequential data write
40 ' save "TELNO1"
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
120 OPEN "O",#1,"1:TELNODAT"
130 INPUT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ(ｵﾜﾘ=99)";A$
140 IF A$="99" THEN GOTO 200
150 INPUT "      TEL No ｦ ﾄﾞｳｿﾞ";B$
160 PRINT #1,A$;",";B$
170 C=C+1
180 IF C=5 THEN C=0 : CLS
190 GOTO 130
200 CLOSE #1
300 '
310 CLS
320 LOCATE 10,6 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 15,8 : PRINT "( y/n ) ?"
340 Y$=INKEY$
350 IF Y$="" THEN GOTO 340
360 IF Y$="Y" OR Y$="y" THEN RUN "MENU02"
370 END
```

変更したところは、上の画面でスミアミの部分です。あとはすべてそのままです。

まだ実行しなくてけっこうです。データ読み出しの節で合わせて行うことにしましょう。セーブを忘れないように……。

```
SAVE "TELNO1" ↵
```

3・3 データの追加

この部分も、前章の「EITAN4」を手直しして使えそうですね。

この節では、プログラムを一部修正して流用する場合の方法をまじえて説明を進めることにします。

プログラムを画面に表示して修正する場合、モードは80桁×25行のほうがやりやすいので、まず次のようにキーインします。

```
WIDTH 80,25 ⏎
```

次に、ベースになるプログラムをロードします。「LOAD"」は (PF6) キーに定義されていますので、これを使うと便利ですね。

```
LOAD "EITAN4" ⏎
```

さて、これで前準備ができました。いよいよプログラムの修正にとりかかりますが、その前にもう1つやっておきたいことがあります。

プログラムの修正では、リストを部分的に画面に表示するケースが多くなります。

LISTコマンドが (PF2) キーに定義されているのはみなさんごぞんじのとおりですが、このキーを押した場合には、

LIST ⏎

と同じ、つまり、いつもプログラムの先頭からリストが表示されてしまいます。そこで次のように、(PF2) キーの定義を変更します。

```
KEY 2,"LIST " ⏎
```

こうすることにより、私たちは (PF2) キーを用いて任意の行を画面に表示させることができます。

ではさっそく修正作業にとりかかりましょう。

画面をクリアしてから次のようにキーインしてください。

(PF2) (スペース) [1] [0] [-] [1] [9] [0] ⏎

次ページに示すプログラムリストはスミアミのところがすでに修正ずみですが、みなさんの画面と形は同じだと思います。

いいですか？ それではカーソル移動キーを使って次ページの画面の修正個所にカーソルをもっていき、画面を見ながら修正してください。(カナ)キーの切り換えをしっかりしてください。インサートモー

```
LIST 10-190

10 ' file name ==> TELNO2
20 ' 1984.6.10
30 ' sequential data append
40 ' save "TELNO2"
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
120 GOSUB 1000
130 DIM AA$(10),BB$(10)
140 OPEN "A",#1,"1:EITANDAT"
150 FOR I=1 TO 5
160    INPUT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ (ｵﾜﾘ=99)";AA$(I)
170    IF AA$(I)="99" AND I=1 THEN GOTO 220
180    IF AA$(I)="99" AND I<>1 THEN N=I-1
            : GOSUB 2000 : GOTO 220
190    INPUT "         TEL No ｦ ﾄﾞｳｿﾞ";BB$(I)

Ready
```

ドも必要ですね。

修正できたら ⏎ キーを押してください。カーソルが「Ready」の下で点滅します。これで、この部分修正は終りです。

同様にして、360 行、1040行、1110行、1120行のステートメントについて修正を加えてください。

次に修正後のリストを示しますが、緑色の部分が変更したところです。

```
360 IF Y$="Y" OR Y$="y" THEN RUN "MENU02"
```

```
1040 OPEN "I",#1,"1:TELNODAT"
```

```
1110 LOCATE 7,7:PRINT "ﾏﾂﾋﾞ ﾉ ﾃﾞｰﾀ==> ";A$
1120 LOCATE 7,8:PRINT "ﾄｳﾛｸ ｿｳｽｳ   ==>";C;" ﾒｲ"
```

これですべてが終了しました。「ＥＩＴＡＮ４」を部分修正して新しいプログラム「ＴＥＬＮＯ２」を作ることができたのです。

それでは、忘れないうちにこのプログラムをセーブしておきましょう。

```
SAVE "TELNO2" ↵
```

最後にプログラム総リストを示します。

〈注〉2200行と2240行のＳＰＣ関数の出力値を「35」に変更してください。

データ追加プログラムリスト

```
10 ' file name ==> TELNO2
20 ' 1984.6.10
30 ' sequential data append
40 ' save "TELNO2"
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLS
120 GOSUB 1000
130 DIM AA$(10),BB$(10)
140 OPEN "A",#1,"1:EITANDAT"
150 FOR I=1 TO 5
160   INPUT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ (ｵﾜﾘ=99)";AA$(I)
170   IF AA$(I)="99" AND I=1 THEN GOTO 220
180   IF AA$(I)="99" AND I<>1 THEN N=I-1
           : GOSUB 2000 : GOTO 220
190   INPUT "      TEL No ｦ ﾄﾞｳｿﾞ";BB$(I)
200 NEXT I : N=I-1
210 GOSUB 2000 : GOTO 150
220 CLOSE
300 '
310 CLS
320 LOCATE 10,6 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 15,8 : PRINT "( y/n ) ?"
340 Y$=INKEY$
350 IF Y$="" THEN GOTO 340
360 IF Y$="Y" OR Y$="y" THEN RUN "MENU02"
370 END
1000 '
1010 ' last data read
1020 '
1030 LOCATE 10,5 : PRINT "ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ"
1040 OPEN "I",#1,"1:TELNODAT"
```

```
1050 IF EOF(1) THEN GOTO 1090
1060 INPUT #1,A$,B$
1070 C=C+1
1080 GOTO 1050
1090 CLOSE #1
1100 PRINT
1110 LOCATE 7,7:PRINT "ﾏﾂﾋﾞ ﾉ ﾃﾞｰﾀ==> ";A$
1120 LOCATE 7,8:PRINT "ﾄｳﾛｸ ｿｳｽｳ  ==>";C;" ﾒｲ"
1130 PRINT
1140 LOCATE 12,10:INPUT "OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ";R$
1150 CLS
1160 RETURN
2000 '
2010 ' data correct subroutine
2020 '
2030 CLS
2040 LOCATE 5,1 : PRINT "ﾆｭｳﾘｮｸ ﾃﾞｰﾀ ｲﾁﾗﾝ"
2050 FOR J=1 TO N
2060   LOCATE 2,2*J : PRINT 2*J-1;"   ";AA$(J)
2070   LOCATE 2,2*J+1:PRINT 2*J;"   ";BB$(J)
2080 NEXT J
2090 LOCATE 2,13
2100 INPUT "ｼｭｳｾｲ ｼﾏｽｶ (y/n)";ANS$
2110 IF ANS$="N" OR ANS$="n" THEN GOTO 2250
2120 LOCATE 2,13
2130 INPUT "ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ (ｵﾜﾘ=99)";NO
2140 IF NO=99 THEN GOTO 2250
2150 K=(NO+1) ¥ 2
2160 IF NO MOD 2=0 THEN GOTO 2210
2170 LOCATE 2,14
2180 PRINT AA$(K);" ==> "; : INPUT AA$
2190 IF AA$<>"" THEN AA$(K)=AA$
2200 LOCATE 2,14 : PRINT SPC(35) : GOTO 2120
2210 LOCATE 2,14
2220 PRINT BB$(K);" ==> "; : INPUT BB$
2230 IF BB$<>"" THEN BB$(K)=BB$
2240 LOCATE 2,14 : PRINT SPC(35) : GOTO 2120
2250 FOR J=1 TO N
2260   PRINT #1,AA$(J);",";BB$(J)
2270 NEXT J
2280 CLS : RETURN
```

3・4　登録・追加プログラムの実行

この節では、すでに作成ずみの電話番号書き込み用・追加用のプログラムファイルを実行してテストをしてみます。また、この結果作られたデータファイル「TELNODAT」は次節以降で使うことにいたしましょう。

まず、ドライブ0のディスクにはメニュープログラムが保存されていますから、これをロードして実行するところから始めます。

```
RUN "MENU02" ⏎
```

80ページのようなディスプレイ画面になりましたか？

いま私たちはデータの書き込みをしようとしているわけですから、仕事番号は「1」をキーインします。そして ⏎ です。

すると、入力に対しての確認を求めてきますから、もしよければ y をキーインし ⏎ キーを押します。

すぐにデータ書き込みのプログラム「TELNO1」がロードされ、続いて実行されます。画面がクリアされてデータ入力をうながすメッセージが表示されますから、テストデータを入力してみましょう。

次の4名について入力してみましょう。

```
トクガ゛ワ イエヤス  ==>  000-000-0000
トヨトミ ヒデ゛ヨシ  ==>  111-111-1111
オダ゛ノブ゛ナガ゛   ==>  222-222-2222
アケチ ミツヒデ゛    ==>  333-333-3333
```

4人の氏名と電話番号を打ち込み、5人目のメッセージが表示されたら「99」を入力し ⏎ キーを押します。

```
ナマエ ヲ ト゛ウソ゛(オワリ=99)? トクガ゛ワ イエヤス
     TEL No ヲ ト゛ウソ゛? 000-000-0000
ナマエ ヲ ト゛ウソ゛(オワリ=99)? トヨトミ ヒデ゛ヨシ
     TEL No ヲ ト゛ウソ゛? 111-111-1111
ナマエ ヲ ト゛ウソ゛(オワリ=99)? オダ゛ ノブ゛ナガ゛
     TEL No ヲ ト゛ウソ゛? 222-222-2222
ナマエ ヲ ト゛ウソ゛(オワリ=99)? アケチ ミツヒデ
     TEL No ヲ ト゛ウソ゛? 333-333-3333
ナマエ ヲ ト゛ウソ゛(オワリ=99)? █ ←──「99」を入力しリターンキーを押す
```

次に画面には「ほかの仕事をするかどうか」をたずねるメッセージが表示されますから、「y ⏎」と入力します。するとすぐにメニュー

画面に戻ります。

今度は**追加処理メニュー2を選択**してみましょう。1を選択したときと同じ手順で次の画面表示に至ります。このようになっていれば、書き込みは順調にいっているはずです。

```
        デ゛ータ ノ ツイカ ヲ シマス

  マツビ゛ ノ デ゛ータ==> アケチ ミツヒデ゛
  トウロク ソウスウ  ==> 4  メイ

      OK ナラ RETキー ヲ ド゛ウゾ゛? ■
```

どうですか? 間違いなくできていましたか?

それでは ⏎ キーを押して次へ進みましょう。

以下の操作は2・8節で勉強ずみですので、それほど苦もなくできると思います。次の6人の電話番号を追加しておきましょう。

サンプルデータの電話番号はきれいにそろっていますが、これは見やすくするためです。何桁でもかまいません。

```
アライ ハクセキ     ==> 444-444-4444
カイハ゛ラ エツケン  ==> 555-555-5555
カイホ セイリョウ    ==> 666-666-6666
ヨシタ゛ ケンコウ    ==> 777-777-7777
サトウ イッサイ     ==> 888-888-8888
ヨシタ゛ ショウイン  ==> 999-999-9999
```

(注)ブレークキーでプログラムを停止させたときには、ダイレクトモードでCLOSEとキーインし、リターンキーを押しておきましょう。この場合はだいじょうぶですが、ファイルがOPENされたままのときもあります。

6人のデータ入力が終りましたら「99」を入力し、「ホカノシゴトヲシマスカ」というメッセージに対しては「n」をキーインし、本節でのテストを終ります。データ追加の確認をしたい方は、もう一度RUNさせ、次の画面で BREAK キー(注)を押せば確かめられます。

```
        デ゛ータ ノ ツイカ ヲ シマス

  マツビ゛ ノ デ゛ータ==> ヨシタ゛ ショウイン
  トウロク ソウスウ  ==> 10  メイ

      OK ナラ RETキー ヲ ド゛ウゾ゛?

Break In 1140
Ready
■
```

確かに登録されていました(「10 メイ」を指す)

ここでリターンキーを押さずにブレークキーを押す(「OK ナラ RETキー ヲ ド゛ウゾ゛?」を指す)

3·5 データの検索

ここまで進んできたあなたは、もうだいぶシーケンシャルファイルの扱いに慣れたことと思います。私たちが日常かかわっている実務上のファイルと違ってディスクのデータファイルは、いくら書き込まれたといっても、目に見えないものですから実際に読み出して確認してみるまでは不安が残ります。初めのうちはとくに、書き込みがうまくいったのかどうか、追加は確実にできているのかどうか、ほんとうに心配なものです。

しかし、たまには失敗もあるかもしれませんが、実践を積み重ねていくうちにだんだん自信がもてるようになってくると思います。

さて、この節では、初めてのテーマ**「データの検索」**をマスターすることにしましょう。

読んで字のごとく、データを探すこと、自分のほしいデータをコンピュータに探させ画面に表示させようというわけです。

シーケンシャルファイルとランダムファイルの違いのところで述べましたが、シーケンシャルファイルでは、ある1つのデータだけを直接に読み出すことができません。最初のデータからたんねんに読んでいかなければならないのです。ずいぶん融通がきかないと思いますが、コンピュータがやってくれるのですから、私たちはだまって従うことにいたしましょう。

では、いつものとおり出だしの部分から始めましょう。

(注)行番号110のCLEAR文は、このプログラムをRUNさせたときにOut Of String Spaceのエラーになる可能性もありますので入れておきました。

```
10 ' file name ==> TELNO4
20 ' 1984.6.20
30 ' sequential data search
40 ' save "TELNO4"
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
130 GOSUB 1000   ←データ読み出しルーチンへ
```

ファイル名は、これまでの順序でいくと「TELNO3」ですが、データ検索はメニュー画面で仕事番号4にしてしまいましたので、このようにしました。

行番号130のGOSUB文で1000行に飛ばし、ここでデータファイルからデータを読み出し、検索の準備をします。

ここでのデータ検索の方法ですが、ファイルから読み出したデータを配列変数にしまっておいて、画面のメッセージに従って探したい人の名前がキーボードから入力されたら、それと配列にしまわれている文字列とを比較することによって該当するデータを表示するようにします。

配列の有効な使い方

そこでまず配列を宣言しなければならないわけですが、これもあらかじめ

```
DIM AA$(200),BB$(200)
```

というように、一定の大きさで確保しておいてもよいのですが、もし登録されているデータ数が50だとすると、残りのメモリは何の役にもたっていないわけですから、ずいぶんムダな使い方をしていることになります。

これを防ぐ方法として、まず最初に、配列変数ではなく**普通の文字変数でデータを読みだしてデータ数をカウントし、その数を配列の大きさに設定する**ことです。1000行からのサブルーチンでこの処理をすることにしましょう。

```
1000 '
1010 '  data read sub 1
1020 '
1030 OPEN "I",#1,"1:TELNODAT"
1040 IF EOF(1) THEN GOTO 1080
1050 INPUT #1,A$,B$
1060 C=C+1 ←――――データ数をかぞえる
1070 GOTO 1040
1080 CLOSE #1
1090 RETURN
```

変数Cをカウンタにして、A＄、B＄という文字変数を使って、氏名、電話番号２つでひと組のデータを読み出すたびに１ずつカウントしています。

もしＥＯＦ、つまりファイルの中に読み出すデータがなくなれば、ＲＥＴＵＲＮ文でＧＯＳＵＢ文のすぐうしろに戻りますから、そこで配列の宣言をします。

配列の宣言

```
140 DIM AA$(C),BB$(C)
```

配列の大きさが決まりましたので、もう一度ファイルをＯＰＥＮしてデータを読み出し配列の中にしまわなければなりません。二度手間にはなりますが、このほうがメモリの有効利用という点ではずいぶん

プラスになるのです。

配列へのデータ読み込みもサブルーチンにして、行番号を2000からとします。まず150行にGOSUB文を入れ、次にサブルーチンをコーディングします。

```
150 GOSUB 2000
```

```
2000 '
2010 '  data read sub 2
2020 '
2030 OPEN "I",#1,"1:TELNODAT"
2040 FOR I=1 TO C
2050   INPUT #1,AA$(I),BB$(I)
2060 NEXT I
2070 CLOSE #1
2080 RETURN
```

この場合EOF文を入れる必要はありません。というのはすでに変数Cにデータ数が入っていますから、この数を超えて読み出そうとすることがないからです。

これで一応下準備ができました。次はメインルーチンです。

まず**画面中央に仕事の目的を表示**するようにしましょう。

```
160 '
170 LOCATE 12,7 : PRINT "デ-タ ヲ サガシマス"
180 FOR I=1 TO 5000 : NEXT I  ←時間つぶし
```

次は**電話番号を知りたい人の名前を入力する部分**です。(カナ)キーを押してカタカナで入力しなければなりませんし、カタカナをキーボードから探すのは慣れるまでけっこう手間どるものです。ここではちょっと楽をして姓だけをキーインし、同姓の場合の処理は別に考えることにします。姓を入力して ⏎ キーを押したら、データ探しのサブルーチンへ飛ぶようにします。

```
190 '
200 CLS
210 LOCATE 8,5:PRINT "ドナタ ノ TEL No ヲ サガシマスカ"
220 LOCATE 8,6:PRINT "     ナマエ ヲ ドウゾ"
230 LOCATE 8,7:INPUT "==>";NM$
240 L=LEN(NM$)  ←文字列の長さを求める
250 '
260 GOSUB 3000  ←データ検索サブルーチンへ
```

LEN関数

行番号240にＬＥＮ(レングス)関数があります。これは文字列(ストリング)の長さを求める関数で書式は次のとおりです。

```
L=LEN(A$)
```

Ａ＄という文字列の長さがＬに代入されます。文字列の前後にスペースが入っているときは、そのスペースも長さに加えられることに注意してください。

ここでＬＥＮ関数を使いましたのは、配列ＡＡ＄(Ｉ)に読み込まれている名前はフルネームで入っているため、このうち姓の部分だけを取り出して比較するためです。

つまり、ＡＡ＄(Ｉ)の左からＬによって与えられる長さだけ文字列を取り出せば、それが姓にほかならないわけです。

データ検索のサブルーチンは、これを利用して次のようにプログラミングしました。

```
3000 '
3010 '  data search sub
3020 '
3030 FOR I=1 TO C
3040 IF NM$=LEFT$(AA$(I),L) THEN N=N+1
      : AC$(N)=AA$(I) : BC$(N)=BB$(I)
3050 NEXT I
3060 RETURN
```

LEFT$
(レフトダラー)
RIGHT$
(ライトダラー)
MID$
(ミッドダラー)

配列ＡＡ＄(Ｉ)の中をデータ数Ｃによって与えられる数だけ探せばよいわけですから、ＦＯＲ～ＮＥＸＴループにします。

そして、あなたが入力した名前が登録されているデータの中にあるかどうかの判断はＬＥＦＴ＄文を使った比較によって行っています。

ＬＥＦＴ＄文と同じように、ある一定の長さの文字列を取り出す関数にＲＩＧＨＴ＄関数、ＭＩＤ＄関数があります。

```
A$=LEFT$(B$,L)    ←B$の左からL個の文字列をA$に代入
A$=RIGHT$(B$,L)   ←B$の右からL個の文字列をA$に代入
A$=MID$(B$,L,M)   ←B$の左からL番目より数えてM個の文字列をA$に代入
```

機能については上のコメントで説明していますので省きますが、けっこう便利な関数です。ダイレクトモードで次ページのようにキーインし、この関数の機能を実際に確かめてみましょう。

文字変数はつなげることもできますから、工夫次第でかなりユニークな使い方ができるのではないかと思います。あなたも、プログラミングの中で役立ててみてはいかがでしょうか？

```
A$="0123456789"

Ready
B$=LEFT$(A$,5) : C$=RIGHT$(A$,5) : ? B$,C$
01234          56789

Ready
D$=MID$(A$,4,3) : ? D$
345

Ready
█
```

さて、3040行のＬＥＦＴ＄関数で登録されている氏名の左からＬ文字をとり出し、それをＮＭ＄と比較し、２つの文字列がもし同じであれば、ＴＨＥＮ文以降のステートメントへ進みます。

電話番号を調べたい人の名前がみつかったわけですから、カウンタＮをプラス１しておきます。変数Ｎは初めは０ですから、１つ見つかれば０＋１＝１、つまり１がＮに代入されます。

次に、**探しあてた人の氏名と電話番号を新しい配列ＡＣ＄とＢＣ＄に代入**しています。この配列は、同姓の人は複数人いる可能性もあるわけですから、同姓の人をすべて退避させておくためのものです。

配列を使った登録方法

たとえば、佐藤さんという人が３人いたとしますと、ファイルに登録してある順に、ＡＣ＄(１)、ＡＣ＄(２)、ＡＣ＄(３)の中に格納されていくわけです。そして、これを再び画面に表示して選択させればよいわけです。

初めの入力のときにフルネームにしておけば、このようなことはあまり起こらないでしょうが、入力が姓のみである上にカタカナの入力ですから、読み方が同じの人はけっこういるかもしれません。

とにかく、プログラミングするときは、いろいろなケースを想定しておくことが大切です。

このサブルーチンからＲＥＴＵＲＮ文によって戻ると、今度は、**見つからなかった場合、１名見つかった場合、複数人の場合の処理をする**ことになります。

この処理過程もやはりサブルーチンにすることにします。

メインルーチンはできるだけ簡素にしておいて、うしろのほうにいろいろなサブルーチンをつけておくことが望ましいと思います。

では、いま説明した３つの場合を処理する部分を次に示します。

```
270 IF N=0 THEN GOSUB 4000 : GOTO 200
280 IF N=1 THEN GOSUB 5000:GOSUB 7000:GOTO 310
290 GOSUB 6000
300 '
310 CLS
320 LOCATE 10,5 : PRINT "ｹﾝｻｸ ｦ ﾂﾂﾞｹﾏｽｶ"
330 LOCATE 10,6 : INPUT "  ( y/n )";Y$
340 IF Y$="Y" OR Y$="y" THEN N=0 : GOTO 200
350 LOCATE 10,8 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 10,9 : INPUT "  ( y/n )";Y$
370 IF Y$="Y" OR Y$="y" THEN RUN "MENU02"
380 END
```

270行: 見つからない
280行: 1名見つかった
290行: 複数人見つかった
370行: メニュープログラムへ

4000行からのサブルーチンは目的の名前が登録されてなかった場合の処理です。サブルーチンから戻ると、200行へ飛ばしてもう一度入力させせます。

5000行からは、1名見つかった場合、6000行からは同姓の人が複数人いた場合の処理です。

キー入力待ちのサブルーチン

●長いプログラムですと何回もキー入力待ちのステートメントが必要になりますので、サブルーチン化しておくと便利です。

7000行からは、キー入力待ちのサブルーチンとしました。短いものですが、複雑なプログラムではかなり使用する機会がありますので、作っておくと便利です。この程度のプログラムではそれほど必要ないかもしれませんが、ぜひ覚えておいてください。

310行からは、その後の作業をどうするか、その意志を求めるものです。メニュープログラムへ戻れるようにしておきました。

ここまででメインルーチンは終りです。このようにメイン部分は短くしておくとよいと思います。

次に各サブルーチンをコーディングしていきましょう。

該当者なしの場合

まず、ファイルに登録されていなかったとき。

```
4000 '
4010 '  data not found
4020 '
4030 CLS
4040 BEEP : BEEP : BEEP
4050 LOCATE 8,5 : PRINT "ｿﾉｶﾀ ﾊ ﾄｳﾛｸｼﾃｱﾘﾏｾﾝ"
4060 LOCATE 8,7
4070 PRINT " Start from the beginning!"
4080 FOR I=1 TO 3000 : NEXT I
4090 RETURN
```

4040行: ブザー音を出して注意します
4080行: メッセージがすぐ消えないようにするタイマー

画面をクリアし、ブザー音とともにメッセージを表示し、注意をうながしています。

この場合は、飛び先へ戻ったら、もう一度入力をしなおします。

次は、**1名該当者がいた場合**です。　　**該当者1名の場合**

```
5000 '
5010 ' data found sub 1
5020 '
5030 CLS
5040 LOCATE 10,5 : PRINT "TEL No.・・・";BC$(N)
5050 LOCATE 8,7
5060 PRINT "Hit RET key if you are OK!"
5070 RETURN
```

これも、問題ないと思います。配列ＢＣ＄(N)の添字Nは、１名しかいないのですから１でもかまいません。

次が３番目のケース、**複数人いた場合**です。　　**該当者が複数人いた場合**

```
6000 '
6010 ' data found sub 2
6020 '
6030 CLS
6040 LOCATE 8,3
6050 PRINT USING "ﾄﾞｳｾｲ ﾉ ﾋﾄ ｶﾞ ##ﾒｲ ｲﾏｽ";N
6060 FOR I=1 TO N
6070 LOCATE 8,4+I
6080 PRINT USING "## &                    &";I,AC$(I)
6090 NEXT I
6100 LOCATE 10,5+I : INPUT "ﾊﾞﾝｺﾞｳｦ ﾄﾞｳｿﾞ";M
6110 CLS
6120 LOCATE 8,5
6130 PRINT AC$(M);" ｻﾝ ﾉ TEL No. ﾊ"
6140 LOCATE 12,6 : PRINT BC$(M);"  ﾃﾞｽ"
6150 LOCATE 8,8
6160 PRINT "Hit RET key if you are OK!"
6170 GOSUB 7000
6180 RETURN
```

フルネームを表示（6080）
番号を入力（6100）
電話番号表示（6130–6140）

同姓の人が２名以上いた場合には、すべて画面にフルネームで表示し、その中から目的の名前を番号で入力させます。番号をキーインして ⏎ キーを押すと、すぐに電話番号が表示されます。

6160、6170行はキー入力待ちの処理です。⏎キーが押されるまで表示画面がそのまま維持されます。

PRINT USING

6050、6080行にある**PRINT USING文**について簡単に説明しておきます。ごく一般的に使われているのは次の3つです。

```
❶ PRINT USING "#####";A          ←数値を出力する桁数を指定
❷ PRINT USING "#####,";A         ←3桁ごとにコンマでくぎる
❸ PRINT USING "&         &";A$   ←文字を左づめで出力する
```

❶は**出力する数値の桁数を指定**するものです。数値が指定されている桁数（例では5桁）より小さいときは、右づめで出力されます。したがって、縦に並んで出力される数値を右ゾロエにするのによく使われます。

❷は**3桁ごとにコンマが入る**ため、大きな数値を扱うときに見やすくなります。

❸は**左の＆(アンド)マークから右の＆マークまでの字数で（例では11文字）文字列を出力**します。この字数より長ければ余分なものは無視されますし、短かければ左づめで出力されます。

ダイレクトモードでいくつかためしてみてください。

```
A=1234 : B!=123456 : A$="FM-77"

Ready
PRINT USING "#####";A
 1234

Ready
PRINT USING "######,";B!
123,456

Ready
PRINT USING "    &            &";A$
    FM-77

Ready
█
```

（"から＆まで：4スペース、＆から＆まで：12スペース）

（FM-77：スペースが4つ入り5文字目から左づめで出力される）

PRINT USING文にはまだいろいろな使い方がありますが、普通はこの程度で間に合ってしまいます。

ここで紹介しなかったものはマニュアルに載っていますから、使う機会が生じましたら、参照してください。

次は最後のサブルーチンです。

```
7000 '
7010 ' inkey call
7020 '
7030 Y$=INKEY$
7040 IF Y$="" THEN GOTO 7030 ELSE RETURN
```

これで完成しました。プログラムをセーブしてから実行してみましょう。

RUNさせますと次のような画面になります。

```
ト゛ナタ ノ TEL No ヲ サカ゛シマスカ
    ナマエ ヲ ト゛ウソ゛
 ==>? ■
```

姓を入力してみてください。見つからなければ、ピーッとブザー音がして次のメッセージが表示されます。

```
ソノカタ ハ トウロクシテアリマセン

 Start from the beginning!
```

入力画面になりましたら「ヨシダ」とキーインしてください。

```
ト゛ウセイ ノ ヒト カ゛  2メイ イマス

 1 ヨシタ゛ ケンコウ
 2 ヨシタ゛ ショウイン

  ハ゛ンコ゛ウヲ ト゛ウソ゛? ■
```

← 番号を入力する

異常なさそうですね。

最後は次の画面です。

```
ケンサク ヲ ツツ゛ケマスカ
  ( y/n )? n

ホカ ノ シコ゛ト ヲ シマスカ
  ( y/n )? ■
```

← 終りなら大文字か小文字のNを入力

← Yを入力すればメニューへ

```
10 ' file name ==> TELNO4
20 ' 1984.6.20
30 ' sequential data search
40 ' save "TELNO4"
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
130 GOSUB 1000
140 DIM AA$(C),BB$(C)
150 GOSUB 2000
160 '
170 LOCATE 12,7 : PRINT "ﾃﾞｰﾀ ｦ ｻｶﾞｼﾏｽ"
180 FOR I=1 TO 5000 : NEXT I
190 '
200 CLS
210 LOCATE 8,5:PRINT "ﾄﾞﾅﾀ ﾉ TEL No ｦ ｻｶﾞｼﾏｽｶ"
220 LOCATE 8,6:PRINT "      ﾅﾏｴ ｦ ﾄﾞｳｿﾞ"
230 LOCATE 8,7:INPUT "==>";NM$
240 L=LEN(NM$)
250 '
260 GOSUB 3000
270 IF N=0 THEN GOSUB 4000 : GOTO 200
280 IF N=1 THEN GOSUB 5000:GOSUB 7000:GOTO 310
290 GOSUB 6000
300 '
310 CLS
320 LOCATE 10,5 : PRINT "ｹﾝｻｸ ｦ ﾂﾂﾞｹﾏｽｶ"
330 LOCATE 10,6 : INPUT "  ( y/n )";Y$
340 IF Y$="Y" OR Y$="y" THEN N=0 : GOTO 200
350 LOCATE 10,8 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 10,9 : INPUT "  ( y/n )";Y$
370 IF Y$="Y" OR Y$="y" THEN RUN "MENU02"
380 END
1000 '
1010 ' data read sub 1
1020 '
1030 OPEN "I",#1,"1:TELNODAT"
1040 IF EOF(1) THEN GOTO 1080
1050 INPUT #1,A$,B$
1060 C=C+1
1070 GOTO 1040
1080 CLOSE #1
1090 RETURN
2000 '
2010 ' data read sub 2
2020 '
2030 OPEN "I",#1,"1:TELNODAT"
2040 FOR I=1 TO C
2050   INPUT #1,AA$(I),BB$(I)
2060 NEXT I
2070 CLOSE #1
```

```
2080 RETURN
3000 '
3010 ' data search sub
3020 '
3030 FOR I=1 TO C
3040 IF NM$=LEFT$(AA$(I),L) THEN N=N+1
     : AC$(N)=AA$(I) : BC$(N)=BB$(I)
3050 NEXT I
3060 RETURN
4000 '
4010 ' data not found
4020 '
4030 CLS
4040 BEEP : BEEP : BEEP
4050 LOCATE 8,5 : PRINT "ｿﾉｶﾀ ﾊ ﾄｳﾛｸｼﾃｱﾘﾏｾﾝ"
4060 LOCATE 8,7
4070 PRINT " Start from the beginning!"
4080 FOR I=1 TO 3000 : NEXT I
4090 RETURN
5000 '
5010 ' data found sub 1
5020 '
5030 CLS
5040 LOCATE 10,5 : PRINT "TEL No.･･･";BC$(N)
5050 LOCATE 8,7
5060 PRINT "Hit RET key if you are OK!"
5070 RETURN
6000 '
6010 ' data found sub 2
6020 '
6030 CLS
6040 LOCATE 8,3
6050 PRINT USING "ﾄﾞｳｾｲ ﾉ ﾋﾄ ｶﾞ ##ﾒｲ ｲﾏｽ";N
6060 FOR I=1 TO N
6070 LOCATE 8,4+I
6080 PRINT USING "## &                &";I,AC$(I)
6090 NEXT I
6100 LOCATE 10,5+I : INPUT "ﾊﾞﾝｺﾞｳｦ ﾄﾞｳｿﾞ";M
6110 CLS
6120 LOCATE 8,5
6130 PRINT AC$(M);" ｻﾝ ﾉ TEL No. ﾊ"
6140 LOCATE 12,6 : PRINT BC$(M);"  ﾃﾞｽ"
6150 LOCATE 8,8
6160 PRINT "Hit RET key if you are OK!"
6170 GOSUB 7000
6180 RETURN
7000 '
7010 ' call inkey
7020 '
7030 Y$=INKEY$
7040 IF Y$="" THEN GOTO 7030 ELSE RETURN
```

3・6　データの修正

第2章の「英単語練習プログラム」では、シーケンシャルファイルにすでに書き込んだデータの修正は扱いませんでした。

しかし電話番号のファイルとなると、たまには番号変更もあるでしょうし、また姓が変わったりする人もいます。ですから、ある人の名前を入力したら、そのデータを検索して氏名・電話番号ともに、自由に修正することができたら大変便利だと思いませんか？

この章では、この辺を学んでいこうというわけです。

データ修正の手順

シーケンシャルファイルにおけるデータ修正は、まず初めに、現在すでにディスクに書き込まれているデータを、一度すべて読み出し、それを配列にしまっておきます。

そして、ディスプレイ上のメッセージに従って、キーボードから変更したい人の名前を入力します。次に、この入力データをもとに配列の中から探し出し、これをディスプレイに表示させ、正しい氏名、電話番号を入力させ、配列の中身を入れかえてしまいます。

この操作を修正の必要のあるデータ数だけくりかえし、最後に**修正された配列の中身を、データファイルをＯＰＥＮして書き込みます。**これで修正されたマスターファイルの出来上がりです。

もう、お気づきのことと思いますが、データ読み出しのところ、またプログラムの全体的な流れなど、前節のデータ検索とだいぶ共通しています。

そんなわけで、データサーチのプログラムに手を加えたり、あるいは、追加サブルーチンなどを作ったりして完成させていくことにいたしましょう。

まず**前節のプログラムをロードしてください。**そして、できるだけスクリーンエディタでなおしていきましょう。

```
LOAD "TELNO4" ⏎
```

まず初めは、プログラムの見出し部分です。

前節のプログラム名と順序が前後しますが、ファイル名は「ＴＥＬＮＯ３」としました。

```
10 ' file name ==> TELNO3
20 ' 1984.6.20
30 ' sequential data correct
40 ' save "TELNO3"
50 '
```

行番号100から160までの初期設定部分は「ＴＥＬＮＯ４」と同じです。

次はメインルーチンです。

```
170 LOCATE 10,7 : PRINT "ﾃﾞｰﾀ ﾉ ｼｭｳｾｲ ｦ ｼﾏｽ"
180 FOR I=1 TO 5000 : NEXT I
190 '
200 CLS
210 LOCATE 8,5:PRINT "ﾄﾞﾅﾀ ﾉ ﾃﾞｰﾀ ｦ ｼｭｳｾｲ ｼﾏｽｶ"
220 LOCATE 8,6:PRINT "    ﾅﾏｴ ｦ ﾄﾞｳｿﾞ"
230 LOCATE 8,7:INPUT "==>";NM$
240 L=LEN(NM$)
250 '
260 GOSUB 3000
270 IF N=0 THEN GOSUB 4000 : GOTO 200
280 IF N=1 THEN GOSUB 5000 : GOTO 310    ← GOSUB 7000は削除
290 GOSUB 6000 : GOSUB 5000              ← 追加
300 '
310 CLS
320 LOCATE 10,5 : PRINT "ｼｭｳｾｲ ｦ ﾂﾂﾞｹﾏｽｶ"
330 LOCATE 10,6 : INPUT "  ( y/n )";Y$
340 IF Y$="Y" OR Y$="y" THEN N=0 : GOTO 200
345 GOSUB 7000                           ← 追加ステートメント
350 LOCATE 10,8 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 10,9 : INPUT "  ( y/n )";Y$
370 IF Y$="Y" OR Y$="y" THEN RUN "MENU02"
380 END
```

追加変更点は上のリスト中、緑色の部分です。

このプログラムでは、7000行からのキー入力待ちルーチンは必要なくなりましたので削除しました。7000行からの削除は、まだ行わないでけっこうです。

5000行からを修正サブルーチンとし、該当データが１名の場合も複数名の場合も、このルーチンを使うことにしました。280行、290行で5000行へ飛んでいますね？

320、330行で修正を続けるかどうかの確認をし、もし終りにするの

であれば、**データ書き込みのサブルーチン**へ飛ばします。7000行からがそれです。行番号を345として、ＧＯＳＵＢ文を追加しました。

行番号1000および2000からのデータ読み込み用サブルーチンは、まったく同じです。

データ検索サブルーチン

次は3000行からの**データ検索のサブルーチン**です。

```
3000 ′
3010 ′ data search sub
3020 ′
3030 FOR I=1 TO C
3040 IF NM$=LEFT$(AA$(I),L) THEN N=N+1 : X(N)=I
     : AC$(N)=AA$(I) : BC$(N)=BB$(I)
3050 NEXT I
3060 RETURN
```

挿入（: X(N)=I）

この部分での変更点は１か所、3040行に代入文を１つ挿入しました。

インデックス

Ｘ(Ｎ)という配列は一種の**インデックス**で、ＡＣ＄(１)、ＡＣ＄(２)、……の内容が、配列ＡＡ＄の何番目に対応しているかを示すものです。修正はＡＣ＄、ＢＣ＄という配列を使って行いますので、修正後、その修正データをもとの配列ＡＡ＄、ＢＢ＄に戻してやらなければなりません。そのため、**戻す先が何番目かを記憶しておき、代入の橋わたしをするのが配列Ｘです。**

検索サブルーチンで修正すべきデータが見つかったら、いよいよ修正作業をしなければなりません。これは5000行からです。

データ修正サブルーチン

```
5000 ′
5010 ′ data correct sub
5020 ′
5030 CLS
5040 LOCATE 5,2 : PRINT AC$(N);" ｻﾝ ﾉ ｼｭｳｾｲ"
5050 LOCATE 5,3
5060 INPUT "    ﾏﾁｶﾞｲ ﾅｲﾃﾞｽｶ ( y/n )";Y$
5070 IF Y$="N" OR Y$="n" THEN RETURN
5080 LOCATE 5,4
5090 PRINT "ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ(ｼｭｳｾｲ ﾅｼ=RET)"
5100 LOCATE 5,5 : PRINT "ｼﾒｲ= ";AC$(N)
5110 LOCATE 5,6 : INPUT "    ===> ";NM$
5120 IF NM$<>"" THEN AA$(X(N))=NM$
5130 LOCATE 5,7 : PRINT "TEL No= ";BC$(N)
5140 LOCATE 5,8 : INPUT "    ===>";TEL$
5150 IF TEL$<>"" THEN BB$(X(N))=TEL$
5160 RETURN
```

配列AA$、BB$の内容を修正値と入れ換え（5120行、5150行）

このサブルーチンは全面的に変わります。このリストどおりにキーインしてください。前のプログラムの5000行からは削除しなくてもかまいません。あとから入力した行番号のステートメントが優先されますから、自動的に切り換わります。

5060、5070行では、画面に表示された人の修正でよいかどうか確認しています。もし「N」であれば、飛び先へ戻り、初めからやりなおします。

5090行からが修正です。氏名、電話番号とも修正できるようになっています。

もし修正がなければリターンキーを押します。この場合、該当者の配列ＡＡ＄の内容は変化しません。つまり、修正しなかったということになりますね。

修正したければ、画面に表示される矢印の右側から正しいデータをキーインし、次にリターンキーを押します。すると、この修正データが配列ＡＡ＄とＢＢ＄の該当する位置に代入され、前のデータは消えてしまいます。ここでの手助けとなるのが**配列X(N)**です。この配列に、何番目に代入したらよいかその数値がしまわれていましたね。修正が終ったらＲＥＴＵＲＮです。

しかし、まだデータファイルには書き込まれていませんよ。配列の中で修正された、というだけです。

次に、6110行から6180行までは不要ですから削除し、新たに6110行をＲＥＴＵＲＮ文にします。

DELETE

```
DELETE 6110-6180 ⏎
```

```
6000 '
6010 ' data found sub 2
6020 '
6030 CLS
6040 LOCATE 8,3
6050 PRINT USING "ト゛ウセイ ノ ヒト カ゛ ##メイ イマス";N
6060 FOR I=1 TO N
6070 LOCATE 8,4+I
6080 PRINT USING "## &                    &";I,AC$(I)
6090 NEXT I
6100 LOCATE 10,5+I : INPUT "ハ゛ンコ゛ウヲ ト゛ウソ゛";N
6110 RETURN
```

6110 RETURN ← キーインする

MをNに変更しました → N

この部分は、同姓の人が複数人いた場合、どの人のデータを修正するのか決定するサブルーチンです。

次に、7000行からのデータ書き込みのサブルーチンを次のように入力してください。

現在メモリに入っている7000行以降のステートメントは新しいステートメントが上書きされるので心配はありません。

```
7000 '
7010 '  data write sub
7020 '
7030 KILL "1:TELNODAT"                  ←ファイルを削除
7040 OPEN "O",#1,"1:TELNODAT"           ←書き込みモードでファイルをオープン
7050 FOR I=1 TO C
7060   PRINT #1,AA$(I);",";BB$(I)       ←修正データの書き込み
7070 NEXT I
7080 CLOSE #1
7090 RETURN
```

これは一部修正された配列ＡＡ＄、ＢＢ＄に格納されているデータを、あらためて書き込むサブルーチンです。

これでもうおわかりいただけたと思いますが、**シーケンシャルデータファイルでの修正は、一度既存のデータを配列内に読み込み、修正があれば、その配列の中身をなおし、正された配列のデータを再びデータファイルに書き込む**のです。

ただし、書き込み用のデータファイルがすでにディスクに存在するわけですから、**アウトプットモードでオープンする前にＫＩＬＬコマンドで削除**しておかなければなりません。

これでデータ修正のプログラムは完成しましたが、１つだけ書き添えておきます。

それは、このプログラム中でＡＣ＄、ＢＣ＄、Ｘという名の配列を使用していますが、配列宣言をどこでもしていないことです。このプログラムでは同姓の人がそんなにたくさんいるとは思われませんので、配列宣言をしませんでしたが、このプログラムを何か違うテーマに応用したりして、類似項目が10個以上あるような場合には当然、次のようなステートメントを入れておかなければなりません。

```
115 DIM AC$(20),BC$(20),X(20)
```

次にプログラムリストをかかげます。

```
10 ' file name ==> TELNO3
20 ' 1984.6.20
30 ' sequential data correct
40 ' save "TELNO3"
50 '
100 WIDTH 40,20 : CONSOLE 0,20,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
130 GOSUB 1000
140 DIM AA$(C),BB$(C)
150 GOSUB 2000
160 '
170 LOCATE 10,7 : PRINT "ﾃﾞｰﾀ ﾉ ｼｭｳｾｲ ｦ ｼﾏｽ"
180 FOR I=1 TO 5000 : NEXT I
190 '
200 CLS
210 LOCATE 8,5:PRINT "ﾄﾞﾅﾀ ﾉ ﾃﾞｰﾀ ｦ ｼｭｳｾｲ ｼﾏｽｶ"
220 LOCATE 8,6:PRINT "      ﾅﾏｴ ｦ ﾄﾞｳｿﾞ "
230 LOCATE 8,7:INPUT "==>";NM$
240 L=LEN(NM$)
250 '
260 GOSUB 3000
270 IF N=0 THEN GOSUB 4000 : GOTO 200
280 IF N=1 THEN GOSUB 5000 : GOTO 310
290 GOSUB 6000 : GOSUB 5000
300 '
310 CLS
320 LOCATE 10,5 : PRINT "ｼｭｳｾｲ ｦ ﾂﾂﾞｹﾏｽｶ"
330 LOCATE 10,6 : INPUT "  ( y/n )";Y$
340 IF Y$="Y" OR Y$="y" THEN N=0 : GOTO 200
345 GOSUB 7000
350 LOCATE 10,8 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 10,9 : INPUT "  ( y/n )";Y$
370 IF Y$="Y" OR Y$="y" THEN RUN "MENU02"
380 END
1000 '
1010 ' data read sub 1
1020 '
1030 OPEN "I",#1,"1:TELNODAT"
1040 IF EOF(1) THEN GOTO 1080
1050 INPUT #1,A$,B$
```

```
1060 C=C+1
1070 GOTO 1040
1080 CLOSE #1
1090 RETURN
2000 '
2010 ' data read sub 2
2020 '
2030 OPEN "I",#1,"1:TELNODAT"
2040 FOR I=1 TO C
2050   INPUT #1,AA$(I),BB$(I)
2060 NEXT I
2070 CLOSE #1
2080 RETURN
3000 '
3010 ' data search sub
3020 '
3030 FOR I=1 TO C
3040 IF NM$=LEFT$(AA$(I),L) THEN N=N+1 : X(N)=I
      : AC$(N)=AA$(I) : BC$(N)=BB$(I)
3050 NEXT I
3060 RETURN
4000 '
4010 ' data not found
4020 '
4030 CLS
4040 BEEP : BEEP : BEEP
4050 LOCATE 8,5 : PRINT "ｿﾉｶﾀ ﾊ ﾄｳﾛｸｼﾃｱﾘﾏｾﾝ"
4060 LOCATE 8,7
4070 PRINT " Start from the beginning!"
4080 FOR I=1 TO 3000 : NEXT I
4090 RETURN
5000 '
5010 ' data correct sub
5020 '
5030 CLS
5040 LOCATE 5,2 : PRINT AC$(N);" ｻﾝ ﾉ ｼｭｳｾｲ"
5050 LOCATE 5,3
5060 INPUT "   ﾏﾁｶﾞｲ ﾅｲﾃﾞｽｶ ( y/n )";Y$
5070 IF Y$="N" OR Y$="n" THEN RETURN
5080 LOCATE 5,4
5090 PRINT "ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ(ｼｭｳｾｲ ﾅｼ=RET)"
5100 LOCATE 5,5 : PRINT "ｼﾒｲ= ";AC$(N)
5110 LOCATE 5,6 : INPUT "    ===> ";NM$
```

```
5120 IF NM$<>"" THEN AA$(X(N))=NM$
5130 LOCATE 5,7 : PRINT "TEL No= ";BC$(N)
5140 LOCATE 5,8 : INPUT "     ==>";TEL$
5150 IF TEL$<>"" THEN BB$(X(N))=TEL$
5160 RETURN
6000 '
6010 ' data found sub 2
6020 '
6030 CLS
6040 LOCATE 8,3
6050 PRINT USING "ドウセイ ノ ヒト ガ ##メイ イマス";N
6060 FOR I=1 TO N
6070 LOCATE 8,4+I
6080 PRINT USING "## &                    &";I,AC$(I)
6090 NEXT I
6100 LOCATE 10,5+I : INPUT "バンゴウヲ ドウゾ";N
6110 RETURN
7000 '
7010 ' data write sub
7020 '
7030 KILL "1:TELNODAT"
7040 OPEN "O",#1,"1:TELNODAT"
7050 FOR I=1 TO C
7060   PRINT #1,AA$(I);",";BB$(I)
7070 NEXT I
7080 CLOSE #1
7090 RETURN
```

いかがですか？ バグはありませんでしたか？

それではさっそく実行してみましょう。このまま PF3 キーを押してもかまいませんし、メニュープログラムをロードして、選択枝でこのデータ修正の仕事を選んでもかまいません。いずれにしても正常に動くはずです。

●プログラムの間違いのことをバグ（虫）といいます。

最初に次の画面となります。

```
ドナタ ノ データ ヲ シュウセイ シマスカ
     ナマエ ヲ ドウゾ
==>? ■
```

ここでは「ヨシダ」と入力してみましょう。

するとすぐに配列ＡＡ＄の中から「ヨシダ」という姓を探し出し、画面に表示します。どうやら同姓の人が２人いたようです。

```
ﾄﾞｳｾｲ ﾉ ﾋﾄ ｶﾞ  2ﾒｲ ｲﾏｽ

 1 ﾖｼﾀﾞ ｹﾝｺｳ
 2 ﾖｼﾀﾞ ｼｮｳｲﾝ

  ﾊﾞﾝｺﾞｳｦ ﾄﾞｳｿﾞ? ■
```

２番目の「ヨシダ」さんを選んでみましょう。

２ ⏎ とキーインして次の画面です。

```
ﾖｼﾀﾞ ｼｮｳｲﾝ ｻﾝ ﾉ ｼｭｳｾｲ
   ﾏﾁｶﾞｲ ﾅｲﾃﾞｽｶ ( y/n )? y
ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ(ｼｭｳｾｲ ﾅｼ=RET)
ｼﾒｲ= ﾖｼﾀﾞ ｼｮｳｲﾝ
   ===> ?◀──────修正しないのでリターンキーを押した
TEL No= 999-999-9999
   ===>? 123-456-7890■◀─修正後リターンキーを押す
```

氏名は修正しないことにして０をキーインしてからリターンキー。電話番号を変更してみました。キーインしてリターンキーを押すと、次の画面になり、「修正を続けますか」の問いかけに「ｎ」を入力すると、修正データがデータファイルに書き込まれ、他の仕事をするかどうかのメッセージが現れます。

これで終りたければ「ｎ」をキーインしましょう。

```
    ｼｭｳｾｲ ｦ ﾂﾂﾞｹﾏｽｶ
      ( y/n )? n

    ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ
      ( y/n )? n

Ready
■
```

3・7 データを印刷する

私たちはこれまで電話番号データファイルの作成をし、データの追加をしたり修正をしたり、また希望する人の電話番号を知ることにも成功しました。

しかし、これらはすべてディスプレイ上で行ってきましたし、登録データすべてをひと目で見るというわけにはまいりませんでした。

また、データファイル作成中は、プログラムに入力ミスがないかどうかのルーチンもつけてはいますが、キー操作は人間のやることです、けっこう間違いがあるものです。

この節では、登録データをプリンタで印刷し、もう一度この目で確かめてみましょう。

また、いままではディスクのデータを読み出して画面に表示はできましたが、それでもまだ、ディスクにほんとうに書き込まれているという実感がなかなかもてませんでした。しかし、プリンタで実際に印刷されてしまえば、信じざるをえないというものです。

画面にデータを表示

登録してあるデータを印刷する前に、まずディスプレイ画面に表示してみましょう。というのは、画面に表示するにはＰＲＩＮＴ文で、一方、プリンタのほうに出力するにはＬＰＲＩＮＴ文を使えばよいわけですから、画面表示さえできていれば、あとはＰＲＩＮＴの前にＬをつけるだけですから、ほぼ出来上がったも同然だからです。それに、ディスプレイ上のほうが操作しやすいですね。

それでは前節のプログラム「ＴＥＬＮＯ３」をロードしてください。

次に、130行から380行まで、3000行から7090行まで削除してください。3000行からは不要ですし、130行からは作りなおさなければなりません。

```
DELETE 130-380

Ready
DELETE 3000-7090

Ready
■
```

次に、最初の部分にファイル名など多少の変更を加えます。

```
10 ' file name ==> TELNO5
20 ' 1984.6.20
30 ' sequential data print out
40 ' save "TELNO5"
50 '
100 WIDTH 80,20 : CONSOLE 0,20,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
```

新ファイル名

横の桁数を80桁に変更

氏名と電話番号を横に続けて並べますので、これまでの40桁ではちょっときゅうくつです。80桁にしておきましょう。

画面表示までのプロセスですが、まず、1000行からのサブルーチンへデータを読みに行き、ここでデータをカウントして配列を宣言します。次に2000行からのサブルーチンでデータをいま宣言した配列に読み込み、これをディスプレイ上に表示すればよいわけです。

メイン部分は次のように作りました。

```
130 '
140 '  main routine
150 '
160 LOCATE 23,3:PRINT "++++ ﾃﾞｰﾀ ﾉ ｲﾝｻﾂ ++++"
170 LOCATE 23,4:PRINT "  5ﾒｲ ｽﾞﾂ ﾋｮｳｼﾞ ｼﾏｽ"
180 GOSUB 1000
190 DIM AA$(C+5),BB$(C+5)
200 GOSUB 2000
210 GOSUB 3000
220 LOCATE 25,5 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
230 LOCATE 25,6 : INPUT "    ( y/n )";Y$
240 IF Y$="Y" OR Y$="y" THEN RUN "MENU02"
250 END
```

配列設定のためのデータ数読み出し（180行）

配列にデータを読み込む（200行）

画面表示のサブルーチンへ（210行）

190行の配列宣言で、その大きさを従来とちがってプラス５していますが、これはデータをディスプレイに表示する場合、５つずつを１グループとしていますので、このようにしました。

というのは、データがいつも５で割り切れる数、たとえば30とか45とかで終っていればよいのですが、はんぱな数18とか47とかで終っていた場合、画面表示をする際、配列ＡＡ＄、ＢＢ＄に、この数を超える添え字が入りますので、次のエラーとなってしまうからです。

```
Subscript Out Of Range In 3090
```

次に画面表示部分のプログラムを示します。

これもサブルーチンにしました。

```
3000 '
3010 '  print out sub
3020 '
3030 E1$="## &←―15スペース―→&"
3040 E2$=" TEL  &←―14スペース―→&"
3050 FOR I=1 TO C STEP 5
3060   FOR J=1 TO 5
3070     M=I+J-1 ←―配列の添字の計算
3080     LOCATE 15,5+J
3090     PRINT USING E1$;M,AA$(M);
3100     PRINT USING E2$;BB$(M)
3110   NEXT J
3120   LOCATE 23,12
3130   PRINT "  Hit any key for next!"
3140   Y$=INKEY$ : IF Y$="" THEN GOTO 3140 ←―何かキーが押されると次の行を実行
3150 NEXT I
3160 CLS : RETURN
```

変数ＪのＦＯＲ～ＮＥＸＴループで５名の氏名と電話番号を表示し、変数ＩのＦＯＲ～ＮＥＸＴループで次のパターンへ移ります。

STEP

５名ずつの表示ですから変数Ｉのきざみは５、ＳＴＥＰ５をつければよいですね。この場合Ｉは、1、6、11、……というように５きざみで変化していきます。そしてこの間の落ちている数、2、3、4、5、7、8、9、10、……などは、ループ変数Ｊの値を加えて調整し、1、2、3、4、5、6、7、……、10、11、……という連続した整数列を作り出しています。この値は3070行で計算され、Ｍに代入されます。このＭの値を配列変数ＡＡ＄、ＢＢ＄の添字に使うわけですね。

〈注〉3030行、3040行の文字型変数Ｅ１＄、Ｅ２＄は、ＰＲＩＮＴ ＵＳＩＮＧ文で使用する編集用です。このプログラムではあまり使いませんが、ＵＳＩＮＧ文を使っての出力並びが同じものがたくさんあるときは、このように変数に代入しておくと便利です。

3140行はキー入力待ちで、何かキーが押されるまでは、この行でとどまっています。

つまり、どんなキーでもよいですから、キー入力があるとこの行を抜けて3150行のＮＥＸＴ文に行き、次の５名が表示されるわけです。

PRINT USING

画面の表示には、前に説明したＰＲＩＮＴ ＵＳＩＮＧ文を使っています。

＆と＆の間のスペースの数は、氏名ではなく住所などですともっとふやさなければならないでしょう。スペースの数とか、表示する位置、つまりＬＯＣＡＴＥ文の座標などは、実際にプログラムを実行させて、調整するとよいでしょう。

さあ、これで完成です。1000行および2000行からのサブルーチンは削除しませんでしたから、ちゃんと残っていますね。リストコマンドで確認してみてください。

だいじょうぶでしたら、このプログラムをセーブしておきましょう。

```
SAVE "TELNO5" ↵
```

ここで、さっそく実行してみたいのですが、その前にもう少しデータを入れておきましょう。

メニュープログラムをロードしてRUNしてください。

```
RUN "MENU02" ↵
```

メニュー画面が現れましたら、2番の「TEL No ノ ツイカ」を選択します。

画面のメッセージに従って、次のデータを入力してください。

```
ｻﾙﾄﾋﾞ ｻｽｹ・・・・・・・・・・000-111-2222
ｷﾘｶﾞｸﾚ ｻｲｿﾞｳ・・・・・・333-444-5555
ﾐﾔﾓﾄ ﾑｻｼ・・・・・・・・・・・666-777-8888
```

入力が終りましたら、「ホカ　ノ　シゴト　ヲ　シマスカ」のメッセージに対して「y」をキーインし、メニュー画面に戻ります。

そこで5番の「TEL LIST ノ インサツ」を選びます。

やがて、このプログラムが実行され、次の画面が現れます。

```
      ++++ ﾃﾞｰﾀ ﾉ ｲﾝｻﾂ ++++
         5ﾒｲ ｽﾞﾂ ﾋｮｳｼﾞ ｼﾏｽ

1 ﾄｸｶﾞﾜ ｲｴﾔｽ          TEL  000-000-0000
2 ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ          TEL  111-111-1111
3 ｵﾀﾞ ﾉﾌﾞﾅｶﾞ          TEL  222-222-2222
4 ｱｹﾁ ﾐﾂﾋﾃﾞ           TEL  333-333-3333
5 ｱﾗｲ ﾊｸｾｷ            TEL  444-444-4444

          Hit any key for next!
```

これは、データファイルの中の最初の5名が表示されたところです。ここで ↵ キーを押しますと次の5名、もう一度押すと先ほど書き込んだ3名の電話番号が表示されます。



どうですか？　ちゃんと書き込まれていましたか？

ここでリターンキーを押しますと、ほかの仕事をするかどうかのメッセージが表示されます。とりあえず「n」とキーインしましょう。

データをプリンタに出力

これで一応データを画面に表示することはできました。次はプリンタに出力して印刷してみましょう。

この操作は、ここまでできていればきわめて簡単で、画面表示のためのＰＲＩＮＴ文をＬＰＲＩＮＴ文に変えるだけでよいのです。しかし、せっかく画面に表示したのですから、これはそのまま生かし、さらにＬＰＲＩＮＴ文を追加することにしましょう。

ＬＰＲＩＮＴ文の入れる場所ですが、ＰＲＩＮＴ文のすぐうしろ、3100行と3110行の間に挿入すればよいでしょう。

次のステートメントを追加しますが、これはスクリーンエディタによって簡単にできます。わざわざキーインする必要はありません。

```
3105     LPRINT USING E1$;M,AA$(M);
3106     LPRINT USING E2$;BB$(M)
```

まず、3090行の３の上にカーソルを重ね、[3] [1] [0] [5]とキーインし、次にＰＲＩＮＴのＰの上にカーソルを重ね、(INS) キーを押し、インサートモードにしてからＬをキーインし [↵] キーを押して終りです。

3106行は同じように3100行を修正すればよいですね。

このスクリーンエディタによって、3090行、3100行は消えることはありませんからご心配なく。

行番号が少しふぞろいになりました。3105行以降を10番きざみになおしておきましょう。

```
RENUM 3110,3105
```

これで完成です。簡単だったでしょう。

それではRUNさせてみましょう。次のように打ち出されてきましたか？

```
 1 トクガ゛ワ イエヤス         TEL  000-000-0000
 2 トヨトミ ヒデ゛ヨシ         TEL  111-111-1111
 3 オダ゛ ノブ゛ナガ゛         TEL  222-222-2222
 4 アケチ ミツヒデ゛           TEL  333-333-3333
 5 アライ ハクセキ             TEL  444-444-4444
 6 カイハ゛ラ エッケン         TEL  555-555-5555
 7 カイホ セイリョウ           TEL  666-666-6666
 8 ヨシダ゛ ケンコウ           TEL  777-777-7777
 9 サトウ イッサイ             TEL  888-888-8888
10 ヨシダ゛ ショウイン         TEL  999-999-9999
11 サルトビ゛ サスケ           TEL  000-111-2222
12 キリガ゛クレ サイゾ゛ウ     TEL  333-444-5555
13 ミヤモト ムサシ             TEL  666-777-8888
14                             TEL
15                             TEL
```

50音順に印刷する

前項で電話番号のリストを無事印刷することができましたが、これは私たちがキーボードから入力し、登録した順になっていて、氏名が50音順になっていません。これを50音順に並べかえて印刷すれば、何かと使い勝手もよくなるのではないかと思います。

ソート

このように**順序を並べかえることをソート**と呼び、ビジネスの分野では、数値を大きい順に並べたり、小さい順に並べたり、とにかく、よく使われます。

私たちも、ここでソートにチャレンジしてみましょう。

この電話帳リストでのソートは、数値ではなく文字を並べかえることになりますが、原理は同じですので、また、そのほうが理解しやすいと思いますので、数値を例にとって説明します。

次に簡単な**ソートプログラム**をかかげます。

〈注〉下のプログラムはサンプルですから入力しなくてもかまいません。キーインする方はＮＥＷコマンドを実行してから入力してください。

```
100 '  sorting
110 '
120 FOR I=1 TO 5
130   READ A(I)            ← ソートするデータを配列に読み込む
140   PRINT A(I)"   ";     ← ソート前のデータを画面に表示
150 NEXT I
160 PRINT                  ← これを入れておかないとソート後のデータが続いて表示されてしまう
170 '
180 FOR I=1 TO 4
190   K=I
200   FOR J=I+1 TO 5
210     IF A(J)<A(K) THEN K=J   ← 小さい値があればポインタKをその値のところに合わせる
220   NEXT J
230   SWAP A(I),A(K)       ← ポインタKの指し示す配列A(K)の内容と配列A(I)の内容を入れかえる
240 NEXT I
250 FOR I=1 TO 5
260   PRINT A(I)"   ";     ← ソート後のデータを画面に表示
270 NEXT I
280 '
290 DATA 4,1,5,3,2         ← ソートするデータ
```

このソートは**バブルソート**と呼ばれるもので、もっとも基本的なものです。

方法は、まずソートするデータを配列に読み込み、ある１つのデータを基準にして、次々と他のデータと大きさを比較していきます。

たとえば、小さい順に並べるのでしたら、基準となっている数より小さい数が現れたら、今度はその数を基準にし、次のデータと比較していきます。

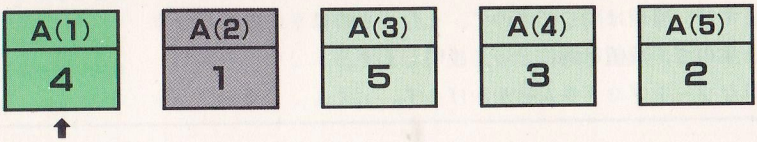
このような操作を次々とくりかえし、最後に基準になっているデータを先頭のほうへもってきます。

この比較する基準となるデータは、ポインタによって指し示されています。上の例でいいますと、**Kがポインタ**になっています。 **ポインタ**

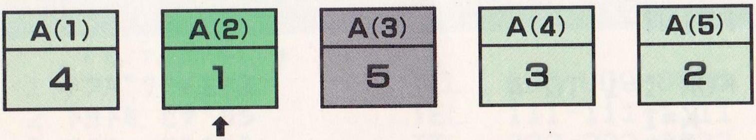
小さいものを先頭にもってくる作業はＳＷＡＰ文で行います。 **SWAP**

230行のＳＷＡＰ文の意味は、Ａ(Ｉ)とＡ(Ｋ)の内容を入れかえるということです。上のプログラムにそって、もう少し説明を加えます。

最初のループ、つまりＩ＝１でＪのループに入ったときの状態は次のようになっています。Ｋ＝1ですから**ポインタは矢印の所**にあります。



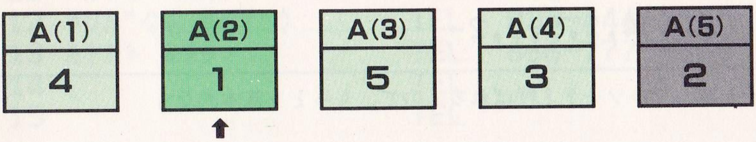
ここで**Ａ(１)とＡ(２)の内容を比較**します。４＞１、つまりポインタが指し示すＡ(１)の内容より小さい値がでてきたわけですから、ポインタをその小さいほうの値のところに合わせます。この場合２番目ですからＫ＝２になります（210行のＫ＝Ｊの操作をしたわけです）。ポインタが移動して次のような状態になりました。



次に、ポインタが指し示す**Ａ(２)の内容とＡ(３)の内容を比較**します。この場合は１＜５ですからポインタは動きません。

次に、やはりポインタが指し示す**Ａ(２)の内容とＡ(４)の内容を比較**します。この場合も１＜３ですから、ポインタは相変わらずＡ(２)のところを指し示しています。

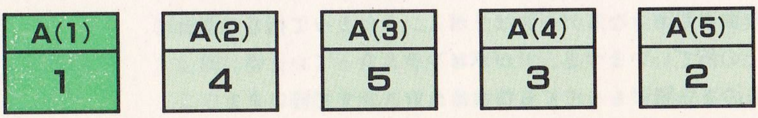
最後、現在のポインタ位置**Ａ(２)の内容とＡ(５)の内容を比較**します。このときの状態は次のようになっています。



当然１＜２ですから、ポインタの位置は変わりません。

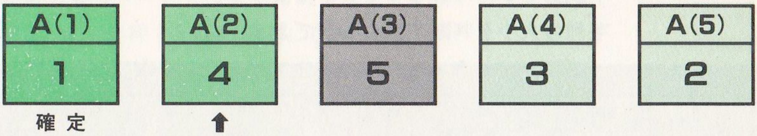
これでＩ＝１のときのＪのループを１回終えたわけです。ですから次のＩのループに行く前に、最小値、つまりポインタが指し示している配列の内容をＡ(１)の内容と入れかえなければなりません。**この操作を230行のＳＷＡＰ文でやっています。**

入れかえた状態は次のようになり、Ａ(１)の内容は５つの数の中の最小値と確定しました。



次にＩ＝２のループでは、最初のポインタＫはＫ＝Ｉですから２にセットされ、図のような状態になり、ポインタの指し示すＡ(２)の内容とＡ(３)の内容を比較することから始まります。

以下、Ｉ＝１のループの説明と同じ操作がくりかえされます。



このループでは、ポインタは最終的にはＡ(５)を指し示すことになり、**Ａ(２)の内容とＡ(５)の内容がＳＷＡＰ文により入れかわります。**

117ページのサンプルプログラムを実行すると次のようになります。

```
RUN
 4     1     5     3     2 ←ソート前のデータ
 1     2     3     4     5 ←ソート後のデータ

Ready
■
```

ソートの原理はわかっていただけましたか？

次に本項のテーマ、**氏名を50音順に並べかえるソート**に入りますが、文字であっても仕組みはまったくかわりません。

数字の場合ですと、その大小はすぐ判断できますが、文字の場合にはどうなっているのでしょうか。

たとえば「ア」のほうが「カ」よりも前にあることは確かですが、このことをコンピュータは何で判断しているのでしょうか？

みなさん、**キャラクタコード**というのをごぞんじですね。

ユーザーズマニュアルの**「キャラクタコード表」**を見てください。

「ア」が＆ＨＢ１、ンが＆ＨＤＤ、10進数になおして177から221までとなっています。この数字の大小で、どちらの文字が先にあるかわかるわけです。つまり、文字のソートも数字のソートに帰するわけです。

```
? ASC("ア"),ASC("カ"),ASC("ラ")
 177            182            215
  ↑              ↑              ↑
  └─────── キャラクタコード ──────┘
Ready
■
```

ASC関数

前ページの画面のようにダイレクトモードでキーインし、テストしてみてください。ここでは**ＡＳＣ関数（アスキー）**を使っていますが、これは、**「文字」を対応するキャラクタコード（10進表記）に変換する**機能があります。

画面をごらんになるとわかるとおり、キャラクタコードによって文字列の大小を判断することができますね。

では、ソーティング部分のサブルーチンを次に示します。

```
4000 '
4010 '  data sort sub
4020 '
4030 FOR I=1 TO C
4040   K=I
4050   FOR J=I+1 TO C
4060     IF AA$(J)<AA$(K) THEN K=J
4070   NEXT J
4080   SWAP AA$(I),AA$(K) : SWAP BB$(I),BB$(K)
4090 NEXT I
4100 RETURN
```

電話番号も同時に入れかえる

〈注〉117ページのサンプルプログラムを入力した方は、ここで「ＴＥＬＮＯ５」をロードしてから上のリストをキーインしてください。

117ページのサンプルプログラムと比べていただくと、形がだいたい同じですから、わかると思います。

ただ、ＳＷＡＰ文で氏名を入れかえるとき、**電話番号も同時に入れかえる**ことを忘れないでください。

なお、文字列の２文字目以降の比較もコンピュータの内部ルーチンが自動的に処理してくれます。

次に、印刷形式に２つのオプションを持たせることにします。つまり、**登録順と50音順に印刷する**方法です。このためのステートメントを170と180行の間に入れておきます。

```
172 LOCATE 27,6 : PRINT "1 ﾄｳﾛｸ ｼﾞｭﾝ"
174 LOCATE 27,7 : PRINT "2 50ｵﾝ ｼﾞｭﾝ"
176 LOCATE 25,9 : INPUT "ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ";P
178 IF P=2 THEN FLAG=1
```

２を選択したらフラグを立てる

フラグのセット

176行のＩＮＰＵＴ文で２番を選択した場合に、178行で**フラグをセット**（１にする）しています。このフラグによって、どちらを選んだかわかるわけです。次のステートメントを追加します。

```
205 IF FLAG=1 THEN GOSUB 4000
```

フラグが1にセットされていれば、ソーティングサブルーチンへ飛ぶことになります。

ソートする、しないにかかわらず、印刷部分はまったく同じです。

〈注〉フラグとは、ある条件判断の目じるしにするもので、変数名は何を使ってもかまいません。

やっと、この節のプログラミングも終りました。実行させてみましょう。(PF3) キーを押すと次の画面になります。

```
++++ テ゛ータ ノ インサツ ++++
5メイ ス゛ツ ヒョウシ゛ シマス

1 トウロク シ゛ュン
2 50オン シ゛ュン

ハ゛ンコ゛ウ ヲ ト゛ウソ゛? ■
```

ここでは2をキーインし、50音順に印刷してみましょう。

いかがですか？　ソートされていますか？

```
 1 アケチ ミツヒテ゛        TEL  333-333-3333
 2 アライ ハクセキ          TEL  444-444-4444
 3 オタ゛ ノフ゛ナカ゛      TEL  222-222-2222
 4 カイハ゛ラ エッケン      TEL  555-555-5555
 5 カイホ セイリョウ        TEL  666-666-6666
 6 キリカ゛クレ サイソ゛ウ  TEL  333-444-5555
 7 サトウ イッサイ          TEL  888-888-8888
 8 サルトヒ゛ サスケ        TEL  000-111-2222
 9 トクカ゛ワ イエヤス      TEL  000-000-0000
10 トヨトミ ヒテ゛ヨシ      TEL  111-111-1111
11 ミヤモト ムサシ          TEL  666-777-8888
12 ヨシタ゛ ケンコウ        TEL  777-777-7777
13 ヨシタ゛ ショウイン      TEL  999-999-9999
14                          TEL
15                          TEL
```

このプログラムのセーブはもうしましたか？　忘れないようにしてください。

次にプログラムリストをのせておきます。

```
10 ' file name ==> TELNO5
20 ' 1984.6.20
30 ' sequential data print out
40 ' save "TELNO5"
50 '
100 WIDTH 80,20 : CONSOLE 0,20,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
130 '
140 '  main routine
150 '
160 LOCATE 23,3:PRINT "++++ ﾃﾞｰﾀ ﾉ ｲﾝｻﾂ ++++"
170 LOCATE 23,4:PRINT "  5ﾒｲ ｽﾞﾂ ﾋｮｳｼﾞ ｼﾏｽ"
172 LOCATE 27,6 : PRINT "1 ﾄｳﾛｸ ｼﾞｭﾝ"
174 LOCATE 27,7 : PRINT "2 50ｵﾝ ｼﾞｭﾝ"
176 LOCATE 25,9 : INPUT "ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ";P
178 IF P=2 THEN FLAG=1
180 GOSUB 1000
190 DIM AA$(C+5),BB$(C+5)
200 GOSUB 2000
205 IF FLAG=1 THEN GOSUB 4000
210 GOSUB 3000
220 LOCATE 25,5 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
230 LOCATE 25,6 : INPUT "    ( y/n )";Y$
240 IF Y$="Y" OR Y$="y" THEN RUN "MENU02"
250 END
1000 '
1010 '  data read sub 1
1020 '
1030 OPEN "I",#1,"TELNODAT"
1040 IF EOF(1) THEN GOTO 1080
1050 INPUT #1,A$,B$
1060 C=C+1
1070 GOTO 1040
1080 CLOSE #1
1090 RETURN
2000 '
2010 '  data read sub 2
```

```
2020 '
2030 OPEN "I",#1,"TELNODAT"
2040 FOR I=1 TO C
2050   INPUT #1,AA$(I),BB$(I)
2060 NEXT I
2070 CLOSE #1
2080 RETURN
3000 '
3010 ' print out sub
3020 '
3030 E1$="## &               &"
3040 E2$=" TEL  &              &"
3050 FOR I=1 TO C STEP 5
3060   FOR J=1 TO 5
3070     M=I+J-1
3080     LOCATE 15,5+J
3090     PRINT USING E1$;M,AA$(M);
3100     PRINT USING E2$;BB$(M)
3110     LPRINT USING E1$;M,AA$(M);
3120     LPRINT USING E2$;BB$(M)
3130   NEXT J
3140   LOCATE 23,12
3150   PRINT "  Hit any key for next!"
3160   Y$=INKEY$ : IF Y$="" THEN GOTO 3160
3170 NEXT I
3180 CLS : RETURN
4000 '
4010 ' data sort sub
4020 '
4030 FOR I=1 TO C
4040   K=I
4050   FOR J=I+1 TO C
4060     IF AA$(J)<AA$(K) THEN K=J
4070   NEXT J
4080   SWAP AA$(I),AA$(K) : SWAP BB$(I),BB$(K)
4090 NEXT I
4100 RETURN
```

注意——1

本節のプログラムを実行すると、「50音順」か「登録順」かのメニュー画面が表示されますが、どちらかを選択して登録リストの画面表示および印刷に移ってもメニュー画面がすぐには消えません。この辺がちょっと不愉快だという方は、次のプログラムを入力してください。この問題は解決します。

```
177 LOCATE 27,6 : PRINT SPC(20)
178 LOCATE 27,7 : PRINT SPC(20)
179 LOCATE 25,9 : PRINT SPC(20)
180 IF P=2 THEN FLAG=1
185 GOSUB 1000
```

注意——2

みなさんは、これまで作ったプログラムを「コピー版のシステムディスク」にセーブしてあるだけではありませんか？

ちょっとしたことがキッカケでディスクをこわしてしまうことがよくあります。これまでのプログラムはあまり長くありませんから、もう一度キーインしなおしてもたいしたことはありませんが、長いプログラムになるとウンザリします。

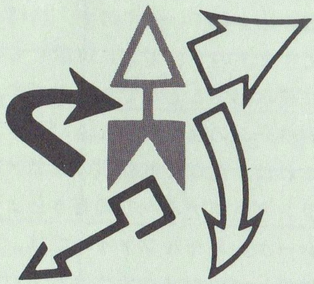
そこでもう一枚ディスクを用意し、いま使っているディスクをそっくりコピーしておくとよいでしょう。

方法は1・5節で説明したからわかりますね。

以後、新しいプログラムを作ってセーブするときは、この2つのディスクにセーブしておくことをおすすめします。

第4章

ランダムファイル[I]



4・1　ランダムファイルについて

ランダムファイルとはいったいなんでしょうか。

私たちが前章までに勉強したシーケンシャルファイルは、あるデータを読み出したいといっても直接読み出すことはできませんでした。検索にしろ、修正にしろ、そのデータファイルに書き込まれているデータを、好き嫌いにかかわらず、初めから読み込んでいかなければなりませんでした。

要するに、長さのまちまちのデータが連続的に書き込まれていたわけですから、書き込まれたデータの並びには規則性がありません。

これに対してランダムファイルは、データが一定の決められた長さで書き込まれていますので、当然、データの書き込まれ方に規則性がでてきますので、どこに位置しているかということさえわかれば、直接に読み出すことができるのです。

右ページの図をごらんください。

これは、前章の電話帳の氏名・電話番号のシーケンシャルファイルとランダムファイルでの書き込まれ方を図式化したものです。

シーケンシャルファイルでは、各項目が、コンマで区切られているだけで、ビッシリつめこまれています。△がありますが、これは姓と名を分けるために必要なスペースです。

文字は１文字が１バイトで書き込まれますから（ただし濁点は１文字分ですから「ガ」などは２バイト必要ですが、説明上ここでは１バイトと考えます)、「トクガワ　イエヤス」は９バイト、「オダ　ノブナガ」は７バイトというように、長さは一定しておりません。

可変長レコード

このようにシーケンシャルファイルではレコードの長さが決まっていませんので**「可変長レコード」**と呼ばれ、ディスクの使用効率も非常によいのですが、ファイルの何番目に位置しているかは、一度データを全部読み出してみなければわかりません。

一方ランダムファイルは、図のように氏名および電話番号のうしろにムダなスペース（△印）がたくさんあいておりますが、いかにも整然と並んでいます。

図の場合ですと、どのデータも氏名に17バイトが、電話番号に21バイトが割り当てられています。シーケンシャルファイルでしたら「トクガワ　イエヤス」の場合、氏名と電話番号で９＋12で21バイトですむところをランダムファイルでは17バイトもよけいにムダな使い方をしているのです。

しかし、メリットもおおいにあります。各データの書き込まれる長

トクガワ△イエヤス	,	000−000−0000	,	トヨトミ△ヒデヨシ	,	111−1	
11−1111	,	オダ△ノブナガ	,	222−222−2222	,	アケチ△ミツヒデ	
	,	333−333−3333	,				
	,	キリガクレ△サイゾウ	,	333−444−			
5555	,	ミヤモト△ムサシ	,	666−777−8888			

⬆シーケンシャルファイル

ランダムファイル⬇

トクガワ△イエヤス△△△△△△△△△	000−000−0000△△△△△△△△△
トヨトミ△ヒデヨシ	111−111−1111
オダ△ノブナガ	222−222−2222
アケチ△ミツヒデ	333−333−3333
キリガクレ△サイゾウ	333−444−5555
ミヤモト△ムサシ	666−777−8888

さが一定していますので（ランダムファイルのレコードは「固定長レコード」といいます）、インデックスにより、誰々さんは何番目に書き込まれているということがわかれば、シーケンシャルファイルのように全部読み出さなくても直接にその個所だけ読み出すことができます。

固定長レコード

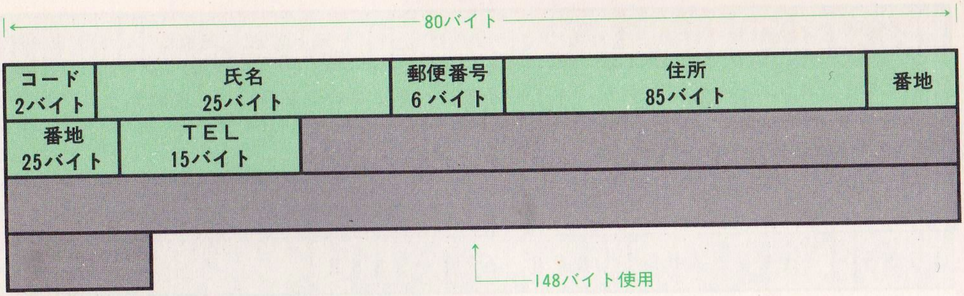
第2章第2節で簡単にふれましたがディスクには**サイド、トラック、セクタ**という単位がありました。**ランダムファイルでの読み書きは、一番小さな単位、つまりセクタごとに行われます。**そして1セクタは256バイトです。

セクタ

1セクタ＝256バイト

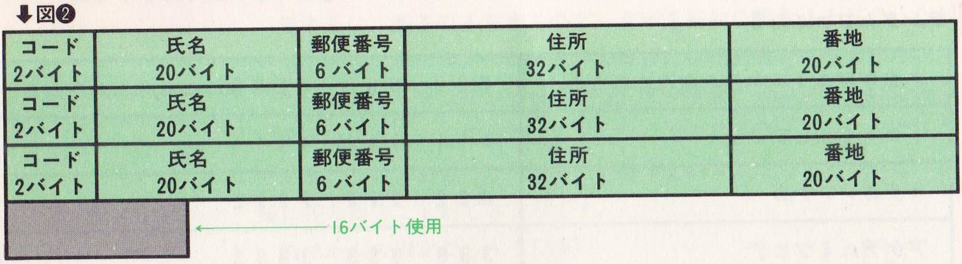
ですから、この256バイト以内で各項目に決められた(固定された)長さを割り当てなければなりません。

次ページの図をごらんください。図❶、図❷とも、1セクタ256バイトにDM（ダイレクトメール）用の各項目を割り当てたものですが、図❶には1名分、図❷では3名分入っています。図❶の場合は、ずいぶん未使用の部分が多くムダな使い方をしていますし、一方図❷では



↑図❶

↓図❷



余りが16バイトで大変効率的です。

このくらいの人数ですと、メモリのムダづかいの様子はまだ実感としてわかりませんが、たとえば100人のＤＭ（ダイレクトメール）用データファイルを作る場合、書き込みは256バイト（１セクタ）単位で行われますから、図❶の場合100セクタ必要になってきます。ところが図❷ですと、34セクタあれば十分です。66セクタも差がでてきます。

このように、１セクタ内でフォーマットをどう決めていくかによって、ランダムファイルにおいても、シーケンシャルファイルと同じようにディスクを有効に活用できるのです。

こう考えますと、慣れてしまえば、ランダムファイルのほうがずっと便利に使えますし、いや、ビジネスの分野ではどうしてもランダムファイルでなければ用をなさないといったほうがよいかもしれません。

以上、ランダムファイルについての概略を述べましたが、少しわかりにくかったかもしれませんが、わからなければそのままにして、少しずつ具体例を学習していきましょう。やがて、その仕組が理解できると思います。

コンピュータは理屈、理論の面もおおいにありますが、とくにソフトのレベルでは、実際にやってみて、体で覚えてしまうのが一番早いようです。

●項目、レコード
上の図の「コード」「氏名」「郵便番号」「住所」「番地」などの１つ１つを「項目」といいます。そして、これらをひとまとめにしたものを「レコード」と呼びます。

ファイルをOPENする

ランダムファイルの場合もシーケンシャルファイルと同じように、データファイルに何か書き込んだり、あるいは読み出したりするとき、**ファイルをＯＰＥＮしなければなりません。**

これは、前にも説明しましたが、本とかノートの内容を見るときにページを開かなければならないのと同じです。

書式は、シーケンシャルファイルと形は同じですが、「モード」指定が今度「R」になります。

OPEN

```
OPEN "R",#1,"1:DMAILDAT"
```

"1:DMAILDAT"→ドライブ番号
#1→ファイル番号
"R"→ランダムファイルモード

ファイルディスクリプタの扱いはシーケンシャルファイルのときと同じです。ドライブ０の場合にはドライブ番号は省略できます。

ファイル番号（使用するバッファ番号）は、ディスク起動時の

```
How many disk files(0-15)? ■
```

で指定した数を超えてはいけません。

また、シーケンシャルファイルではＩＮＰＵＴモード、ＡＰＰＥＮＤモードの場合には、ＯＰＥＮしたときディスクにファイルがないと、

```
File Not Found
```

のエラーメッセージがでましたが、ランダムファイルの場合はＯＰＥＮ文だけではエラーは出ません。

このＯＰＥＮ文でファイルを開いてはじめて、データを読んだり書いたりすることが可能になります。

ファイルをCLOSEする

ファイルをＯＰＥＮして、使い終ったら、行儀よくＣＬＯＳＥすることにしましょう。これは、シーケンシャルファイルの場合とまったく同じです。

CLOSE

```
CLOSE #1,#2
```

#1, #2→ファイル番号

ファイル番号が省略されたときは、そのときＯＰＥＮしているファイルがすべてＣＬＯＳＥされます。ファイル番号は上記のように２つ

でも３つでもかまいません。コンマで区切って複数指定できます。

データ書き込みの前準備

ファイルをＯＰＥＮしたということは、「そのファイルをこれから使いますよ」と宣言したことにはなりますが、このままでは書き込みはできません。前準備が必要なのです。

128ページでダイレクトメール用のデータの割り当てを行ったように、１セクタ（256バイト）の中を、**どんな項目でどんな割りつけをして使うのか、その形を決めておかなければいけない**のです。

氏名には何バイト、住所には何バイト、というように定義しておく必要があるのです。この定義は**ＦＩＥＬＤ（フィールド）文**によって行いますが、ランダムファイルでは、この決められたフォーマットに従って、どのレコードも同じ様式で書き込まれます。

ＦＩＥＬＤ文

ＦＩＥＬＤ文の書式は次のとおりです。

```
FIELD #1,15 AS A1$,20 AS A2$,50 AS A3$
```

- 1 … ファイル番号
- 15, 20, 50 … フィールド幅（バイト数）
- A1$, A2$, A3$ … 文字変数

この例ですと、ファイル番号１番（というより**１番バッファ**）に文字変数Ａ１＄、Ａ２＄、Ａ３＄にそれぞれ15バイト、20バイト、50バイトのフィールド幅を割り当てたことになります。ですから、実質使用フィールド幅は85バイトになり、残り（256－85）の171バイトは使われないことになります。

フィールド幅の合計が256バイトになるまで、どのように割り当ててもかまいません。そして**割り当てを行う変数は必ず文字変数を使います**（扱うデータが数値であっても、です）。

ＦＩＥＬＤ文によって、バッファ内に変数のエリアが決まりましたが、書き込む前にもう１つ準備が必要です。

それは、このＦＩＥＬＤ文の定義に従って、書き込むデータをバッファ上にセットしなければなりません。つまり、ＦＩＥＬＤ文で使われている変数に、それぞれ対応するデータをはめこむのです。

ＬＳＥＴ（エルセット）
ＲＳＥＴ（アールセット）

この、**データをバッファ上にセットする方法**に次の２つがあります。

```
❶ LSET A1$=X$
❷ LSET A2$=Y$
❸ RSET A3$=Z$
```

- ❶ バッファ上のＡ１＄にＸ＄の内容をセット
- ❷ バッファ上のＡ２＄にＹ＄の内容をセット
- ❸ バッファ上のＡ３＄にＺ＄の内容をセット

❶も❷もともにバッファ上の変数に、X＄、Y＄、Z＄に入っている文字列をセットすることに変わりはないのですが、❶の場合はLEFT、つまり左づめにセットされ、❷の場合はRIGHT、つまり右づめにセットされる点が違います。いずれの場合も、セットしようとした文字列がフィールド幅より長いときは余分な文字は無視されます。

次に、フィールドは文字変数で定義されていますから、数値データをセットするときには、これを文字型に変換しなければなりません。数値が整数か、単精度か、倍精度かによって、次のように変換方法が異なります。

●MKI＄（メイクインテジャードル）、MKS＄（メイクシングルドル）、MKD＄（メイクダブルドル）はそれぞれ、整数型、単精度型、倍精度型の数値を文字列に変換する機能をもっています。

```
LSET B1$=MKI$(X%)    ←整数データの変換
LSET B2$=MKS$(Y!)    ←単精度データの変換
LSET B3$=MKD$(Z#)    ←倍精度データの変換
```

この場合LSETになっていますが、RSETでもかまいません。

MKI＄
MKS＄
MKD＄

ここで1つ大切な注意があります。整数データ、単精度データ、倍精度データに対応するバッファ上の文字変数（上の例ではB1＄、B2＄、B3＄）はフィード幅が決まっていて、それぞれ2バイト、4バイト、8バイトとなっています。

数値データのFIELD設定

ですから、数値データを書き込むときは、数字の型によって次のようにFIELDを定義しておかなければなりません。

```
FIELD #1,2 AS B1$,4 AS B2$,8 AS B3$
```

- 2 AS B1＄：整数型の場合
- 4 AS B2＄：単精度型の場合
- 8 AS B3＄：倍精度型の場合

これでデータ書き込みの下準備ができました。

データを書き込んでみよう

シーケンシャルファイルではPRINT＃文を使って書き込みをしましたが、ランダムファイルではPUT（プット）文を使います。書式は次のとおりです。

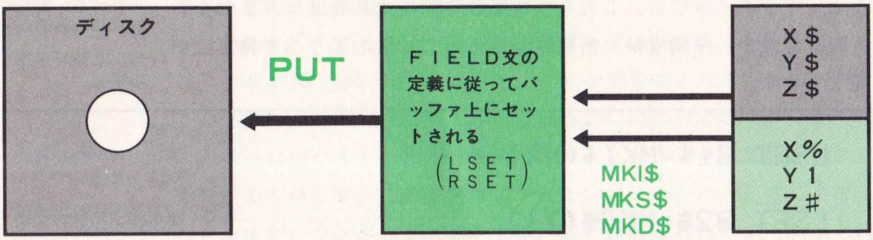
PUT

```
PUT #1,S
```

- 1：ファイル番号
- S：セクタ番号

ファイル番号で指定されたバッファの内容（256バイト分）をＳで示されるセクタ番号のところへ書き込む（ＰＵＴする）ということになります。セクタ番号は省略することもできますが、その場合、直前に書き込みが行われたセクタの次にＰＵＴされます。

ここまでのプロセスをまとめると次図のようになります。



ＭＫＩ＄、ＭＫＳ＄、ＭＫＤ＄によって文字型に変換された数値データ、および一般の文字変数が、ＦＩＥＬＤ文で決められたフィールド幅にそれぞれバッファ上でセットされ（ＬＳＥＴ、ＲＳＥＴ）、このセットされたバッファの内容）(256バイト、１セクタ分）がＰＵＴ文によってディスクに書き込まれます。

次に示すのは書き込みのテストプログラムです。

```
100 ' test program for random file
110 '
120 OPEN "R",#1,"1:TESTDAT"                    ← ファイルを開く
130 FIELD #1,2 AS CD$,20 AS NM$,15 AS TEL$     ← フィールドを定義
140 '
150 INPUT "ｺｰﾄﾞ No.(END=9999)";CD              ┐
160 IF CD=9999 THEN GOTO 270                   ├ データ入力
170 INPUT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ";A$                     │
180 INPUT "TEL No. ｦ ﾄﾞｳｿﾞ";B$                 ┘
190 S=S+1          ← 1レコードの入力ごとに書き込みセクタを1ずらす
200 '
210 LSET CD$=MKI$(CD)   ┐
220 LSET NM$=A$         ├ バッファ上でデータをセット
230 LSET TEL$=B$        ┘
240 PUT #1,S            ← ディスクにデータを書き込む
250 '
260 GOTO 150            ← ループ
270 CLOSE #1            ← ファイルを閉じる
280 END
```

プログラムは複雑ではありませんから、ごらんいただけばわかると思いますが、書き込みまでの流れは次のようになっています。

データ書き込みの流れ

❶ ファイルを開く（ＯＰＥＮする）
❷ フィールドを設定する（ＦＩＥＬＤ文）
❸ データを入力する
❹ バッファ上にデータをセットする
❺ ディスクにデータを書き込む

これがデータ書き込みの一般型です。このパターンさえ覚えてしまえば、あとはディスクをどう有効利用するか、ファイルの形をどうするかなどの課題があるだけで、それほどむずかしくはありません。

コードは４桁の数値で入力することにしてありますので、210行でバッファ上にセットする場合、ＭＫＩ＄を使っています。

書き込むセクタ番号は、コード、名前、電話番号一式を入力するごとに１ずつ加算しています。

このプログラムでは、コードで２バイト、名前で20バイト、電話番号で15バイトとっていて、これを１レコードとしていますから、全部で37バイト、したがって256から37を引いた残り219バイトは未使用のままです。

プログラムを実行する前に、よくおかす間違いを１つ。

ＦＩＥＬＤ文で次のような設定をするとエラーになります。

```
500 FIELD #1,2 ASC$,20 AS NAME$
```

NAME$ ← 氏名に対するフィールド割り当て
ASC$ ← コードに対するフィールド割り当て

コードと氏名に対してバッファ上のフィールドを設定したわけですが、前者はＡＳとＣ＄の間にスペースを入れ忘れた結果ＡＳＣという予約語と重複したために、後者はすでにＮＡＭＥという予約語があるために、どちらもSyntax Errorになります。

それではプログラムをＲＵＮさせて、次の６名のデータを入力してみましょう。再び、前章の方々に登場してもらいました。

```
0001    ﾄｸｶﾞﾜ ｲｴﾔｽ    000-000-0000
0101    ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ    03-222-0000
1002    ｵﾀﾞ ﾉﾌﾞﾅｶﾞ    045-111-3333
3012    ｱｹﾁ ﾐﾂﾋﾃﾞ     2222-55-4444
0133    ｱﾗｲ ﾊｸｾｷ      03-000-5555
9012    ｶｲﾊﾞﾗ ｴｯｹﾝ    444-66-1111
```

```
ｺｰﾄﾞ No.(END=9999)? 0001
ﾅﾏｴ ｦ ﾄﾞｳｿﾞ? ﾄｸｶﾞﾜ ｲｴﾔｽ
TEL No. ｦ ﾄﾞｳｿﾞ? 000-000-0000
ｺｰﾄﾞ No.(END=9999)? 0101
ﾅﾏｴ ｦ ﾄﾞｳｿﾞ? ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ
TEL No. ｦ ﾄﾞｳｿﾞ? 03-222-0000
ｺｰﾄﾞ No.(END=9999)? 1002
ﾅﾏｴ ｦ ﾄﾞｳｿﾞ? ■
```

上の画面は入力中の様子を示したものです。

コード、氏名、電話番号の１ブロックの入力が終るたびにカチッと音がしてディスクへの書き込みが行われます。６つのデータですから、最後のＥＮＤマーク（9999）が入力されるまでに、６レコード書き込まれたことになります。

```
OPEN "R",#1,"1:TESTDAT"  ←データファイルをオープン

Ready
? LOF(1)                 ←ファイルの長さを調べる
 6                       ←セクタ数

Ready
CLOSE                    ←必ず閉じることを忘れずに！

Ready
■
```

ＬＯＦ関数（レングスオブファイル）

新しい関数がでてきました。ＬＯＦ関数です。

これは、ファイルの長さを知るための関数で、ＰＲＩＮＴ文で上のように入力すると、すぐ下に長さを表示してくれます。上の画面では６となっていますね。

ちょっとわかりかけてきたのではありませんか？

そうです、私たちがいま入力したのは６名分のデータ、レコードにして６レコードだったのです。そしてこの場合１レコードで256バイト、つまり１セクタを使っていましたから、データファイル「ＴＥＳＴＤＡＴ」分として全部で６セクタ、ディスク上を占有していたのです。

ここでもう1つ調べてみたいことがあります。

ドライブ1に入っているデータディスクのファイル名を画面に表示してみてください。FILESコマンドでよかったですね。

```
FILES "1:"
```

```
EITANDAT  1 A S 1
TELNODAT  1 A S 1
TESTDAT   1 A R 1

149 Clusters Free

Ready
```

私たちが　いま書き込んだデータファイルがありますね。そのうしろの数字を見てください。1となっています。

この数字の単位は**クラスタ**といいます。　**クラスタ**

39ページをごらんください。1周、つまり1トラックは16セクタで構成され、その半分、**8セクタを1クラスタといいます。**

ファイルはすべて、このクラスタ単位でディスク上に確保されますから、たとえば、1セクタだけ使っても、8セクタ使っても、やはり1クラスタだけそのファイルのエリアとして占有されてしまうのです。あるファイルをOPENして、1文字だけ書き込んでも、やはり1クラスタとられてしまいます。

ファイルがクラス単位でディスク上にそのエリアを確保することはわかっていただけましたか？

このことをさらに明確にするために、1つ実験をしてみましょう。

次のステートメントを入力してください。

```
145 S=6
```

これはどういうことかといいますと、190行で1ずつ加算するSの値を「6」に初期設定しています。

つまり、7、8、9というようにセクタ番号をふやしているわけですが、すでに6セクタ分書き込まれているわけですから、さらに7、8、9セクタと書き込んでいきますと、8と9の間でクラスタ数が変わるはずです（1クラスタは8セクタでしたね）。

一方、ファイルはクラスタ単位で確保されると書きました。ですから、9セクタ分書き込めば、データファイル「TESTDAT」は2

クラスタを確保することになります。

これさえ確かめられれば、いままでの説明をご理解いただけると思ます。

さあ、それでは、次の3名のデータを入力してください。

```
1111   ｶｲﾎ ｾｲﾘｮｳ     555-111-1111
5000   ﾖｼﾀﾞ ｹﾝｺｳ     333-333-3333
7001   ｻﾄｳ ｲｯｻｲ      222-222-2222
```

いいですか？

入力が終りましたら、再びＦＩＬＥＳコマンドでデータディスクのファル名を画面に表示してみてください。

```
FILES "1:"

EITANDAT   1 A S 1
TELNODAT   1 A S 1
TESTDAT    1 A R 2

148 Clusters Free

Ready
■
```

1クラスタふえている

どうですか？

確かに2クラスタ専有しています。

これで、いままで不明確だった点がだいぶはっきりしたのではありませんか？

ランダムファイルのデータを読む

さて、私たちは９名分のデータをランダムファイルに書き込むのに成功しました。でも、ほんとうに書き込まれているかどうか、ちょっと心配ですね。読み出して画面に表示してみましょう。

ランダムデータを読み出すには、**ＧＥＴ(ゲット)文**を使います。ＧＥＴするのです。なんとなくわかるような感じがするでしょう？

そして書き込みのときにセクタ単位でＰＵＴしたように、読み込むときもセクタ単位でＧＥＴすればよいのです。

ＧＥＴ文

書式は次のとおりです。

```
GET #1,S
```

- S：ファイル内でのセクタ番号
- #1：ファイル番号

バッファ

この書式の意味は、**ファイル番号で指定されたランダムファイルのＳセクタ目のデータを、ファイル番号に対応するバッファ上に読み出す**、ということです。

もう少し具体的にいいますと、上の例は現在ＯＰＥＮしている＃１のランダムファイルの第Ｓセクタを＃１のバッファ上に取り出すということになります。

ファイル番号は１としてありますが、この数は０から15までの数を指定できます。ただし、How many files（０－15）？で入力した数を超えてはいけません。

突然ＧＥＴ文のことを書きましたので、中には、何もしなくてもすぐＧＥＴできるのか、と誤解する方もいるかもしれませんので補足しておきますが、データを読むときも書き込みのときと同じように、**最初にファイルをＯＰＥＮして次にフィールドを設定しておかなくてはなりません。**

そして、このＦＩＥＬＤ文によるフィールドの定義がなされていると、あるセクタをＧＥＴしたときに、自動的に、それぞれ定義されたフィールド幅をもつバッファ上の変数に割り当てられるのです。書き込みのときにはＬＳＥＴまたはＲＳＥＴを使ってセットしなければなりませんでしたが、読むときには必要ないのです。

このようにバッファ上に取り出され、バッファ上の変数に割り当てられたランダムデータを画面に表示するには、文字型データはそのまま使えますが、数値データは書き込みの際に文字型に変換されている

＊ＶＡＬ関数も数値変換に使われますが、ＭＫＩ＄などで文字列に変換してあるものは必ずＣＶＩ、ＣＶＳ ＣＶＤを用いて数値に戻さなければなりません

わけですから、もう一度数値に戻してやらなければなりません。

たとえば、Ｂ１＄、Ｂ２＄、Ｂ３＄というバッファ上の変数に、それぞれ２バイト、４バイト、８バイトの数値が文字型として代入されているとしますと、**ＣＶＩ、ＣＶＳ、ＣＶＤ**＊を用いて次のように数値に戻してやらなければなりません。

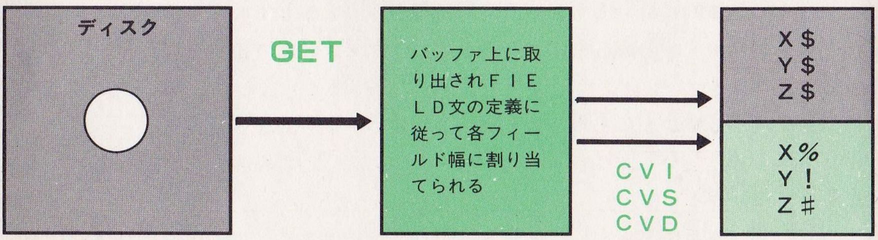
```
X%=CVI(B1$) ←――――― 整数値に変換
Y!=CVS(B2$) ←――――― 単精度型数値に変換
Z#=CVD(B3$) ←――――― 倍精度型数値に変換
```

ＣＶＩ（コンバートインテンジャー）
ＣＶＳ（コンバートシングル）
ＣＶＤ（コンバートダブル）

こうして数値型に変換しておきますと、ＢＡＳＩＣモードで使っていたのと同じ扱いができるのです。

そして最後に、**ファイルを使い終ったらＣＬＯＳＥ**です。

データを読む過程を図にすると次のようになります。



それではデータ読み出しを実行する前に、その手順を箇条書きにして整理しておきましょう。

❶　ランダムファイルを開く（ＯＰＥＮする）。
❷　ＦＩＥＬＤ文でフィールドを定義する。
❸　ＧＥＴ文を用いて１セクタ（256バイト）ずつバッファ上に取りだす。
❹　数値データの場合にはＣＶＩ、ＣＶＳ、ＣＶＤを使って文字型を数値型に変換する。
❺　ランダムファイルを閉じる（ＣＬＯＳＥ）する。

さあ準備完了です。先ほど書き込んだデータを読み出して画面に表示してみましょう。

プログラムを次に示します。行番号を300からとして、データ書き込

み部分のうしろにつけ加えましょう。

```
300 ' random data read
310 OPEN "R",#1,"1:TESTDAT"        ← ファイルをオープン
320 FIELD #1,2 AS CD$,20 AS NM$,15 AS TEL$   ← フィールドの設定
330 FOR I=1 TO LOF(1)      ← 1セクタから最終セクタまでループ
340 GET #1,I
350 PRINT USING "####   ";CVI(CD$);  ┐
360 PRINT NM$;                       │← 画面に表示
370 PRINT TEL$                       │
380 NEXT I                           ┘
390 CLOSE #1         ← ファイルを閉じる
400 END
```

330行のLOF（1）は、1番のファイルの長さ、つまり何セクタ使っているか、その値が与えられます。そして、その値の示す回数だけ340行のGET文によってディスク上のデータがバッファ上に取り出されます。これをPRINT文で画面に表示すればよいわけですが、CD$だけは数値に変換しなければいけませんから、350行でCVIを使って変換しています。 LOF

これを実行するときはRUNコマンドではなく*、次のようにしてください。RUNコマンドですと、このステートメントの前半に書き込み部分がありますから、そちらが実行されてしまいます。

＊もしRUNコマンドで実行してしまったときはストップキーを押してCLOSEをキーインし、再びリターンキーを押しておいてください。

```
GOTO 300 ⏎
```

次のような画面になりましたか？

```
GOTO 300
   1  トクガ゛ワ イエヤス          000-000-0000
 101  トヨトミ ヒデ゛ヨシ          03-222-0000
1002  オダ゛ ノブ゛ナガ゛           045-111-3333
3012  アケチ ミツヒデ゛            2222-55-4444
 133  アライ ハクセキ              03-000-5555
9012  カイバ゛ラ エッケン          444-66-1111
 111  カイホ セイリョウ            555-111-1111
5000  ヨシダ゛ ケンコウ            333-333-3333
7001  サトウ イッサイ              222-222-2222

Ready
█
```

ちゃんと書き込みができていたようですね。

しかし、コード番号の並びが少々ふぞろいですね。

350行でＰＲＩＮＴ　ＵＳＩＮＧを使っていますから桁ぞろえはできているのですが前がデコボコです。これをきれいに整とんするには、次の345行、346行をキーインし、350行を少し変更すればうまくいきます。やってみましょう。

```
345 CD=CVI(CD$) : CO$=MID$(STR$(CD),2)
346 CO$=RIGHT$("000"+CO$,4)
350 PRINT CO$"   ";
```

数値を文字列に変換（STR$(CD)）

4桁の文字列にする（346行）

ＳＴＲ＄(エス・ティー・アール・ドル)

ＳＴＲ＄関数は数値を文字列に変換しますが、数値の前にかくされている符号も含めて変換しますので、つまり１番目はプラスの符号になっていますから、２番目以降を取り出さなければなりません。そしてこれに３桁の文字列「０００」をつなげ、右側から４桁抜き出せば４桁の文字列としてのコードができあがるわけです。

実行して確認してみてください。

ここで１つ実験をします。330行を次のように変えて実行してみてください。

```
330 FOR I=1 TO LOF(1)+1
```

```
7001   ｻﾄｳ ｲｯｻｲ                222-222-2222

Input Past End In 340
Ready
CLOSE

Ready
█
```

エラーメッセージがでてストップしてしまいました。

まず、ＣＬＯＳＥしておいてください。

これは340行で「ファイルの最終レコードを超えて読み出そうとしている」という意味のエラーです。ファイルを扱うようになると、このエラーはよく出会うエラーです。ここではファイル長ＬＯＦ(1)にプラス１しているのですからエラーの原因がわかりますが、ここがＶとかＹとかいった変数だと、しばし考え込むことがよくあります。

4・2 メニュープログラムとデータ書き込み

この節では、前節で得た知識をもう一度確認する意味で、簡単な宛て名書き印刷をやってみます。

データファイルには、次の項目を保存しておきます。

❶ **コード番号**
❷ **氏名**
❸ **郵便番号**
❹ **住所**
❺ **番地**

これらを１レコードとして１セクタ内に書き込みます（と、今は考えて下さい)。少しムダな使い方ですが、これは学習のためですし、慣れるためにはある程度簡素化しておいたほうがよいと思うからです。

宛て名印刷の方法は、宛て名印刷をしたい人のコードを次々と入力し、エンドマークを入れると、あとはプリンタがかってに印刷をしてくれるというふうにします。

コード番号にはいろいろなつけ方があります。下２桁を県別にするとか、職種の区別に第何桁を使うとか、また使用桁数もケースによってだいぶ違いがでてくると思います。しかしこの本では、**データの入出力をどう扱うか、**という点にテーマをしぼっていますのでコードのつけ方にはとらわれずに話を進めることにいたします。

メニュープログラム

メニュー画面は「ＭＥＮＵ０１」を少し手直しして、次のようにしました。

```
        アテナ インサツ プログラム

  データ ノ ニュウリョク・・・・・・・1
  データ ノ シュウセイ・・・・・・・・2
  アテナ ノ インサツ・・・・・・・・・3
  オワリ・・・・・・・・・・・・・・・4

        input job number ? ■
```

メニュープログラムのリストを次に示します。手直しした個所にはコメントを入れておきました。

```
10 ' file name ==> MENU03
20 ' 1984.6.20
30 ' menu program for address printing
40 ' save "MENU03"
50 '
100 WIDTH 40,25 : CONSOLE 0,25,0,0
110 DEFINT A-Z
120 CLS
130 '
140 READ MENU
150 DIM M$(MENU)
160 FOR I=1 TO MENU
170   READ M$(I)
180 NEXT I
190 '
200 LOCATE 7,4 : PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,5:PRINT "|  ｱﾃﾅ ｲﾝｻﾂ ﾌﾟﾛｸﾞﾗﾑ  |"
230 LOCATE 7,6 : PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270   LOCATE 8,7+I : PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13 : INPUT "input job number ";NO
310 LOCATE 8,15 : PRINT M$(NO)
320 LOCATE 18,16 : INPUT "OK (y/n)";Y$
330 IF Y$="N" OR Y$="n" THEN CLS : GOTO 200
340 ON NO GOTO 360,370,380,390
350 '
360 RUN "DMAIL1"
370 RUN "DMAIL2"
380 RUN "DMAIL3"
390 CLS
400 LOCATE 15,7 : PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
410 END
```

ファイルの変更（10行・40行 MENU03）
仕事の変更（30行 address printing）
見直しの変更（220行）
プログラム名の変更（360〜380行）

```
1000 '
1010 '  data area
1020 '
1030 DATA 4
1040 DATA ﾃﾞｰﾀ ﾉ ﾆｭｳﾘｮｸ・・・・・・・1
1050 DATA ﾃﾞｰﾀ ﾉ ｼｭｳｾｲ・・・・・・・・2
1060 DATA ｱﾃﾅ ﾉ ｲﾝｻﾂ・・・・・・・・・3
1070 DATA ｵﾜﾘ・・・・・・・・・・・・・4
```

メニューの変更

データ入力部分のプログラム

プログラムの先頭部分は、いつものようにファイル名、作成年月日を変えたりするだけでよいでしょう。

```
10 ' file name ==> DMAIL1
20 ' 1984.6.20
30 ' random data write
40 ' save "DMAIL1"
50 '
100 WIDTH 40,20 : CONSOLE 0,25,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
```

次にデータ書き込み用のランダムファイルをＯＰＥＮしなければなりませんが、各項目が使用するフィールドを次のように設定することにします。

●フィールドの割当て●

氏名		郵便番号		住所		番地	
変数	バイト	変数	バイト	変数	バイト	変数	バイト
NM$	20	PN$	6	AD$	35	BA$	20

ここで、おやッと気づいたことはありませんか？

そうです、141ページのファイルに書き込む項目にはコードも入っていましたが、この表には見当たりませんね。コードは書き込まないでいいのでしょうか？　この理由はすぐ説明します。

とにかくフィールドが定義できましたから、ランダムファイルをＯＰＥＮする部分をコーディングしてみましょう。

```
130 '
140 ' open file
150 OPEN "R",#1,"1:DMAILDAT"
160 FIELD #1,20 AS NM$, 6 AS PN$,
          35 AS AD$,20 AS BA$
```

インデックス

ランダムファイルにおいては、インデックスという考え方をよく使います。技術関係の本には索引がついていますね。あの索引と同じと考えていただいてよいと思います。

ある専門用語を本文中から探そうとしても、ページをめくっただけではなかなか見つけることができません。ところが索引を調べれば、その用語にページ数が書いてありますから、すぐに探すことができますね。ランダムファイルで使うインデックスも同じことなのです。

たとえば、次の4名のデータを1人を1レコード（1セクタ）としてランダムファイルに書き込むとします。

コード	氏名	郵便番号	住所	番地
0011	ﾄｸｶﾞﾜ ｲｴﾔｽ	101	aaaaaaaaaa	1-2-3
1122	ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ	030	bbbbbbbbbb	456-78
3344	ｵﾀﾞ ﾉﾌﾞﾅｶﾞ	231	cccccccccc	1984ﾊﾞﾝﾁ
5566	ｱｹﾁ ﾐﾂﾋﾃﾞ	680	dddddddddd	9-8-76

このとき、ランダムファイルの考え方では、「コード」「氏名」「郵便番号」「番地」、この5項目を1セクタ内にすべて書き込むのではなく、コードだけはシーケンシャルファイルに書くことにしているのです。

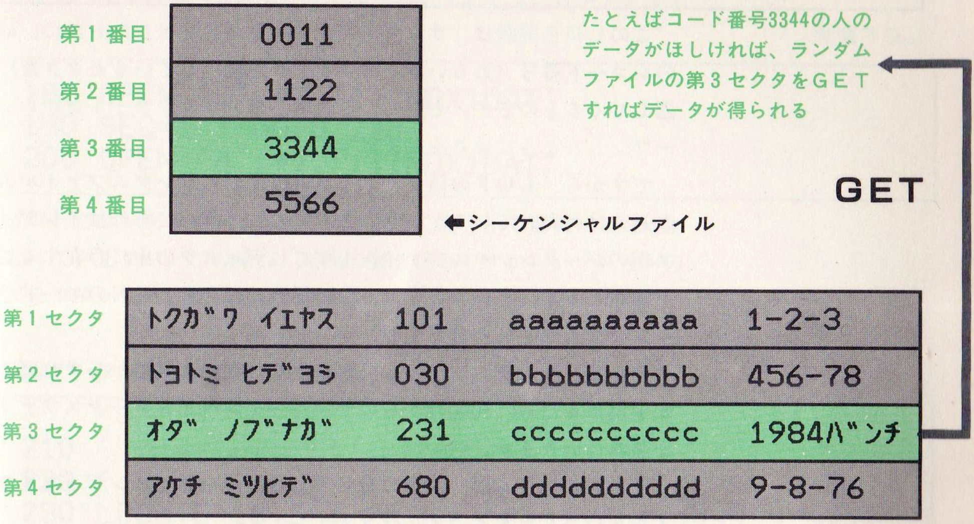
そうしますと、このシーケンシャルファイルに書き込まれたコードが、本でいう索引の役割をするのです。

なぜかといいますと、シーケンシャルファイルは連続的に前のほうから順序よくデータがつめこまれていました。覚えていますか？

つまり、上の例でいいますと、「0011」は1番目に、「1122」は2番目に、「3344」は3番目に、「5566」は4番目に書き込まれます。と同時に、そのコードに対応する「トクガワ」「トヨトミ」「オダ」「アケチ」は、ランダムファイルの第1セクタ、第2セクタ、第3セクタ、第4セクタに書き込まれます。ということは、キーボードからコードを入力したとき、そのコードがシーケンシャルファイルのX番目にあったとしますと、ランダムファイルのXセクタ目をＧＥＴすれば、希望する人

のデータが得られるわけです。

このことを図解したのが次の図です



ご理解いただけたでしょうか？

要するに、データをランダムファイルにセーブするときに、インデックス用のシーケンシャルファイルも一緒に作っておくのです。

さて、メニュープログラムで１番の「データの入力」を選択するとこの「ＤＭＡＩＬ１」が実行されるわけですから、初回の書き込みも複数回目の書き込みも、いずれも考慮してコーディングしなければなりません。

初回でしたら、インデックス用のシーケンシャルファイルはＯＵＴＰＵＴモードでよいですが、２回目、３回目にＯＵＴＰＵＴモードでオープンすると「File Already Exists」のエラーになってしまいます。

OUTPUTモード

それでは常にＡＰＰＥＮＤモードでＯＰＥＮすればよいのでしょうか。一見正しいように思えますが、そうではないのです。

たとえば、初回の入力時にＡＰＰＥＮＤモードでＯＰＥＮすると、

File Not Found

というエラーメッセージが返ってきます。

そこで、初回でも複数回目でも入力可能なようにするには、ちょっとした細工が必要です。次のステートメントを入れておきます。

```
170 IF LOF(1)<>0 THEN GOSUB 1000 : GOTO 200
```

LOF(1)：最大レコードが与えられる

ＬＯＦ関数

このＬＯＦ関数は、ランダムファイルをＯＰＥＮしたときに、最大のレコード番号（あるいはそのファイルが使用しているセクタ数）を返してくるものです。そして、これは、**ランダムファイルを初めてオープンしたときには０を与えます。**

ですから、ＬＯＦの値が０でなければすでにランダムファイルの書き込みは行われているということですし、と同時に、それはインデックス用のシーケンシャルファイルもすでにディスクの中に存在することを意味します。ということは、この場合にはＡＰＰＥＮＤモードでオープンすればよいわけですね。

そして、ランダムファイルの書き込みは何セクタ目からすればよいかを決めるため、1000行からのサブルーチンへインデックスファイルを読みにいきます。

ＬＯＦが０ならば、新規にＯＵＴＰＵＴモードでインデックスファイルをＯＰＥＮすることになります。

1000行からのサブルーチンは次のようになります。

```
1000 '
1010 ' data count sub
1020 OPEN "I",#2,"1:INDX1DAT"
1030 WHILE NOT EOF(2)
1040   INPUT #2,DUMMY
1050   REC=REC+1
1060 WEND
1070 SEC=REC+1
1080 CLOSE #2
1090 RETURN
```

1040：データのカラ読み
1050：データ数をカウントする
1070：総データ数にプラス１した値が入力データを書き込むランダムファイル上のレコード位置になる

ＷＨＩＬＥ〜ＷＥＮＤ

このサブルーチンでは**ＷＨＩＬＥ〜ＷＥＮＤ文**を使っています。これは、ＷＨＩＬＥ以降の条件が満たされているうちは、ＷＨＩＬＥとＷＥＮＤの間をループします。この場合**「２番のファイルが終りでなければループする」**ということになります。1040行でデータの読み飛ばしをして1050行でデータ数を数えています。

1070行のＳＥＣ＝ＲＥＣ＋１は、前ページの図を見ていただけばわかると思いますが、ランダムファイルへの書き込み位置（ＰＵＴする位置）は、シーケンシャルファイルのデータ数プラス１になります。

RETURN文によりサブルーチンから帰った場合は、200行からのAPPENDモードへ、そうでない場合は、初回のデータ入力ということになりますから、OUTPUTモードでインデックスファイルをOPENします。

```
180 OPEN "O",#2,"1:INDX1DAT"
190 SEC=1 : GOTO 220
200 OPEN "A",#2,"1:INDX1DAT"
```

190行でランダムファイルへの書き込み位置を示す変数SECは1となっていますが、この場合、初回の書き込みですから、ファイルの第1セクタからPUTしていきます。

次はデータ入力と書き込み部分です。

```
210 '
220 ' data input
230 LOCATE 10,5 : PRINT "ﾃﾞｰﾀ ﾉ ﾆｭｳﾘｮｸ ｦ ｼﾏｽ"
240 FOR I=1 TO 3000 : NEXT I
250 CLS
260 PRINT "ｺｰﾄﾞ No. ｦ ﾄﾞｳｿﾞ (END=9999)"
270 INPUT "    ==> ";CD
280 IF CD=9999 THEN GOTO 490
290 PRINT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ"
300 INPUT "    ==> ";A1$
310 PRINT "ﾕｳﾋﾞﾝ ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ"
320 INPUT "    ==> ";A2$
330 PRINT "ｼﾞｭｳｼｮ ｦ ﾄﾞｳｿﾞ"
340 INPUT "    ==> ";A3$
350 PRINT "ﾊﾞﾝﾁ ｦ ﾄﾞｳｿﾞ"
360 INPUT "    ==> ";A4$
370 PRINT
380 INPUT "       OK ﾃﾞｽｶ ( y/n )";Y$
390 IF Y$="N" OR Y$="n" THEN GOTO 250
400 '
410 ' data write
420 PRINT #2,CD                          ←インデックスファイルにコードを書き込む
430 LSET NM$=A1$ : LSET PN$=A2$ ┐
440 LSET AD$=A3$ : LSET BA$=A4$ ┘—バッファ上にデータをセット
450 PUT #1,SEC                           ←ランダムファイルにデータを書き込む
460 SEC=SEC+1                            ←次の書き込みのためにセクタ数をふやす
470 GOTO 250
```

データ入力を終りたいときは270行で9999を入力するようにしています。

420行、450行でもわかるように、シーケンシャルファイルは＃２のバッファ、ランダムファイルは＃１のバッファを使っています。１つのバッファを２つ以上のファイルで共用することはできませんから注意してください。

この段階では、ファイルが２つ同時にＯＰＥＮしていますね。

あなたは、ディスク起動時の

```
How many disk files(0-15)? ■
```

というメッセージに対して、どんな数をキーインしましたか？　その数を超えたバッファを同時に使うことはできません。つまり、オープニング時に入力した数字は、バッファ数と考えることもできるのです。

次は、ファイルを閉じてエンディング、そして総リストです。

```
480 '
490 '  file close
500 CLOSE #1,#2
510 '
520 CLS
530 LOCATE 10,4 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
540 LOCATE 10,5 : INPUT "  ( y/n )";Y$
550 IF Y$="Y" OR Y$="y" THEN RUN "MENU03"
560 END
```

```
10 '  file name ==> DMAIL1
20 '  1984.6.20
30 '  random data write
40 '  save "DMAIL1"
50 '
100 WIDTH 40,20 : CONSOLE 0,25,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
130 '
140 '  open file
150 OPEN "R",#1,"1:DMAILDAT"
160 FIELD #1,20 AS NM$, 6 AS PN$,
           35 AS AD$,20 AS BA$
170 IF LOF(1)<>0 THEN GOSUB 1000 : GOTO 200
```

```
180 OPEN "O",#2,"1:INDX1DAT"
190 SEC=1 : GOTO 220
200 OPEN "A",#2,"1:INDX1DAT"
210 '
220 ' data input
230 LOCATE 10,5 : PRINT "ﾃﾞｰﾀ ﾉ ﾆｭｳﾘｮｸ ｦ ｼﾏｽ"
240 FOR I=1 TO 3000 : NEXT I
250 CLS
260 PRINT "ｺｰﾄﾞ No. ｦ ﾄﾞｳｿﾞ (END=9999)"
270 INPUT "    ==> ";CD
280 IF CD=9999 THEN GOTO 490
290 PRINT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ"
300 INPUT "    ==> ";A1$
310 PRINT "ﾕｳﾋﾞﾝ ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ"
320 INPUT "    ==> ";A2$
330 PRINT "ｼﾞｭｳｼｮ ｦ ﾄﾞｳｿﾞ"
340 INPUT "    ==> ";A3$
350 PRINT "ﾊﾞﾝﾁ ｦ ﾄﾞｳｿﾞ"
360 INPUT "    ==> ";A4$
370 PRINT
380 INPUT "       OK ﾃﾞｽｶ ( y/n )";Y$
390 IF Y$="N" OR Y$="n" THEN GOTO 250
400 '
410 ' data write
420 PRINT #2,CD
430 LSET NM$=A1$ : LSET PN$=A2$
440 LSET AD$=A3$ : LSET BA$=A4$
450 PUT #1,SEC
460 SEC=SEC+1
470 GOTO 250
480 '
490 ' file close
500 CLOSE #1,#2
510 '
520 CLS
530 LOCATE 10,4 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
540 LOCATE 10,5 : INPUT "   ( y/n )";Y$
550 IF Y$="Y" OR Y$="y" THEN RUN "MENU03"
560 END
1000 '
1010 ' data count sub
1020 OPEN "I",#2,"1:INDX1DAT"
```

```
1030 WHILE NOT EOF(2)
1040   INPUT #2,DUMMY
1050   REC=REC+1
1060 WEND
1070 SEC=REC+1
1080 CLOSE #2
1090 RETURN
```

プログラムをセーブしたら、下に示すデータを入れて、動作テストをしてみましょう。

バグ

また、途中でエラーがでてストップしたら、ダイレクトで一度ＣＬＯＳＥしてから、チェックしてください。バグ(プログラムの間違い)をとってから再度ＲＵＮする前に、すでにデータファイルがディスクに作られていたら、それをＫＩＬＬコマンドで削除してから始めましょう。正常に動くまでは、めんどうでも、こういったくりかえしが必要です。

ＫＩＬＬ

```
KILL "1:INDX1DAT" ↵

KILL "1:DMAILDAT" ↵
```

```
0011   ﾄｸｶﾞﾜ ｲｴﾔｽ    101   aaaaaaaaaa    1-2-3
1122   ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ   030   bbbbbbbbbb    456-78
3344   ｵﾀﾞ ﾉﾌﾞﾅｶﾞ   231   cccccccccc    1984ﾊﾞﾝﾁ
5566   ｱｹﾁ ﾐﾂﾋﾃﾞ    680   dddddddddd    9-8-76
```

```
ｺｰﾄﾞ No. ｦ ﾄﾞｳｿﾞ (END=9999)
   ==> ? 0011
ﾅﾏｴ ｦ ﾄﾞｳｿﾞ
   ==> ? ﾄｸｶﾞﾜ ｲｴﾔｽ
ﾕｳﾋﾞﾝ ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ
   ==> ? 101
ｼﾞｭｳｼｮ ｦ ﾄﾞｳｿﾞ
   ==> ? aaaaaaaaaa
ﾊﾞﾝﾁ ｦ ﾄﾞｳｿﾞ
   ==> ? 1-2-3■
```

4·3　データの修正

住所の変更とか、入力ミスにあとで気づいたとかした場合、プログラムさえしっかり作っておけば、データの修正は簡単です。

シーケンシャルファイルのときは、氏名の一部を入力して、それと同じ文字列をファイルの中から探して修正を加えていました。カタカナでの入力はちょっとめんどうですし、すでに書き込みずみのデータをすべて配列の中にとり込んでおかなければいけませんでした。

ところがランダムファイルとなると、キー入力はきわめて簡単。数字列のコードをテンキーで入力すればよいのですし、**コードは氏名と1対1の対応をなしています**から、同姓の人を考慮する必要がありません。ただ、コード体系を完成させるまでは少々大変だと思いますが、あとで楽をするためには、初めに苦労はつきものですね。

それに、ディスクに保存できるデータ量もシーケンシャルファイルと比べて格段の差がでてきます。シーケンシャルファイルではすべてのデータを配列に読み込まなければならないので、本体のメモリを考慮すれば、そこにおのずと制限がでてきます。

しかし、ランダムファイルになりますと、シーケンシャルに書き込まれているインデックスは配列にとり込む必要がありますが、その他のデータの入出力は、すべて256バイトのバッファだけで事がすんでしまいます。大変な違いですね。

ちなみに、両面倍密度ディスクの場合、両サイドとも（表、裏面）トラック0、トラック1はＩＰＬ、ディスクコードなどで使用されますが、そのほかはすべてユーザーが使えますので、**38×2×2＝152クラスタがフリー**ということになります。（**1トラック＝2クラスタ**であることを思い出してください。）

これをセクタ数に換算すると、1クラスタ＝8セクタですから、152を8倍して1216セクタとなります。

この章では1セクタに1名のデータしか書き込んでいませんが、もし3名分書き込むとすれば、

1216(セクタ)×3(名)＝3648(名)

の資料をストックできるのです。

これほどのデータ量になりますと、シーケンシャル的な考え方ではとても手におえません。

実用面でデータを管理するには、やはりランダム的な思考法が要求されるのです。

プログラムの作成

この部分はもう、例によって例のごとくといった感じですね。

```
10 ' file name ==> DMAIL2
20 ' 1984.6.20
30 ' random data correct
40 ' save "DMAIL2"
50 '
100 WIDTH 40,20 : CONSOLE 0,25,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
130 '
140 OPEN "R",#1,"1:DMAILDAT"
150 FIELD #1,20 AS NM$, 6 AS PN$,
          35 AS AD$,20 AS BA$
```

ファイル名
ランダムファイルOPEN
フィールドの設定

データを修正する場合には、目的のデータを探し出さなければなりませんから、**インデックス用の配列を作って読み込む**必要があります。それが次のステートメントです。

```
160 '
170 GOSUB 1000 : FLAG=1
180 DIM A(DM) : DM=0 : GOSUB 1000 : INDEX=DM

1000 '
1010 '  index read
1020 '
1030 OPEN "I",#2,"1:INDX1DAT"
1040 WHILE NOT EOF(2)
1050   DM=DM+1
1060   IF FLAG=1 THEN GOTO 1080
1070   INPUT #2,DUMMY : GOTO 1090
1080   INPUT #2,A(DM)
1090 WEND
1100 CLOSE #2
```

170行のＧＯＳＵＢ文で1000行からのサブルーチンへ飛び、インデックスファイルに書き込まれているインデックス(コード番号)の数を読みます。これは、インデックス用の配列の大きさを決めるためです。

配列はあらかじめプログラムの先頭部分で、A(100)、A(500) のよ

うに宣言しておいてもよいのですが、メモリ効率からいくと、170行、180行でのやり方のほうがよいでしょう。

170行のＧＯＳＵＢ文により1000行からのサブルーチンへ飛んだ段階では、まだＦＬＡＧ＝０でフラグは立っていませんから、1070行のＩＮＰＵＴ＃文を実行し、データ数を読みます。そして、ＲＥＴＵＲＮ文で帰ったときにフラグを立て（ＦＬＡＧ＝１）、配列を宣言し、再度同じサブルーチンへ飛び、今度は、いま宣言した配列の中にインデックスを読み込みます。

●プログラミングではフラグという考え方をよく使います。ある条件を満たしたときとか、ある動作を行ったときフラグを立てておけば（１にする）、プログラムの中での条件判断が大変楽です。たとえば、ある家を探すのに目じるしとしてフラグ（旗）を立てておけば探しやすいのと同じです。

２度目にサブルーチンへジャンプする前に、ＤＭ＝０にクリアしておくのを忘れないでください。こうしておきませんと、1050行でさらに加算が行われますから、1080行のＩＮＰＵＴ＃文で、

```
Subscript Out Of Range
```

のエラーになります。つまり、配列がたりなくなってしまった、というわけです。

これで下準備ができましたから、次は修正したいデータの入力部分になります。

```
190 '
200 LOCATE 5,5 : FLAG=0
210 PRINT "ﾄﾞﾅﾀ ﾉ ｼｭｳｾｲ ﾃﾞｽｶ (END=9999)?"
220 LOCATE 7,6 : INPUT "ｺｰﾄﾞ ﾊﾞﾝｺﾞｳ";CD
230 IF CD=9999 THEN GOTO 300
240 LOCATE 10,8 : INPUT "OK ﾃﾞｽｶ ( y/n )";Y$
250 IF Y$="N" OR Y$="n" THEN CLS : GOTO 200
260 GOSUB 2000
270 CLS : GOTO 200
```

コードが入力されると、240行で確認をうながし、間違いなければ260行で、2000行からの修正サブルーチンへ飛びます。

修正作業を終りたいときは、「9999」を入力するようにします。

サブルーチンのプログラムは次のとおりです。

```
2000 '
2010 '  data correct sub
2020 '
2030 CLS
2040 FOR I=1 TO INDEX
2050   IF A(I)=CD THEN FLAG=1:SEC=I:I=INDEX
2060 NEXT I
2070 IF FLAG=0 THEN GOTO 2250
```

入力されたコードをインデックスの中から検索する（2040〜2060行）

コードが見つからないときの処理（2070行）

```
2080 GET #1,SEC ←―――――――――――――――――――――――――――― 該当セクタをGETする
2090 LOCATE 2,3 : PRINT "ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ"
2100 LOCATE 4,4 : PRINT "ｼｭｳｾｲ ﾅｹﾚﾊﾞ RET ｷｰ"
2110 PRINT
2120 PRINT NM$ : INPUT "  ==> ";A1$ ┐
2130 IF A1$="" THEN A1$=NM$         │
2140 PRINT PN$ : INPUT "  ==> ";A2$ │
2150 IF A2$="" THEN A2$=PN$         │
2160 PRINT AD$ : INPUT "  ==> ";A3$ ├―― 正しいデータ入力
2170 IF A3$="" THEN A3$=AD$         │
2180 PRINT BA$ : INPUT "  ==> ";A4$ │
2190 IF A4$="" THEN A4$=BA$         ┘
2200 LSET NM$=A1$ : LSET PN$=A2$ ┐
2210 LSET AD$=A3$ : LSET BA$=A4$ ┴―――― バッファ上にセット
2220 PUT #1,SEC ←―――――――――――――――――――――――― ランダムファイルに書き込む
2230 RETURN
2240 '
2250 BEEP : BEEP : BEEP ←――――――――――――――――――― ブザー音
2260 LOCATE 8,8 : PRINT "ｿﾉｶﾀ ﾊ ﾄｳﾛｸｼﾃ ｱﾘﾏｾﾝ"
2270 FOR I=1 TO 3000 : NEXT I
2280 RETURN
```

2040行と2060行のＦＯＲ～ＮＥＸＴ文で、入力されたコードＣＤと同じものがあるかどうかインデックスを検索します。**もし見つかれば、フラグ(ＦＬＡＧ)を立てておき、その見つかったときのＩの値が、修正すべきレコードが書き込まれているセクタ数になる**わけですから、変数ＳＥＣに代入しておきます。コードとデータは１対１対応ですから、１つ見つかればそれ以上の検索は不要となり、ループ変数Ｉの値に最終値ＩＮＤＥＸを代入し、ＦＯＲ～ＮＥＸＴループから抜け出してしまいます。

2070行でフラグ(ＦＬＡＧ)が立っていなければ、該当データがなかったわけですから、2250行で**ブザー音を鳴らし、画面にメッセージを表示**します。

2080行では、該当セスタをＧＥＴし、修正の準備をします。

データの修正方法は、現在書き込まれているデータを画面に表示し、そのままでよければリターンキーを押し、直したければキー入力するようにしました。

2200行以降で、正しいデータをバッファ上にセットし、ランダムデータファイルに書き込みます。

最後は、使用ファイルのＣＬＯＳＥです。

```
280 '
290 CLOSE #1
300 '
310 CLS
320 LOCATE 10,4 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 10,5 : INPUT "    ( y/n )";Y$
340 IF Y$="Y" OR Y$="y" THEN RUN "MENU03"
350 END
```

ここでちょっと横道にそれた話を１つ。

これまで、画面に表示されたメッセージに対しての答えは、ほとんどの場合、340行のように「ｙ」または「ｎ」でキー入力させていました。そしてキー入力後リターンキーを押さなければいけませんでしたね。

次の２つのプログラムを比べてください。

```
500 INPUT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ (y/n)";Y$
510 IF Y$="Y" OR Y$="y" THEN RUN "MENU"
520 PRINT "ｺﾞｸﾛｳｻﾏ ﾃﾞｼﾀ"
530 END
```

```
500 PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ (y/n)"
510 Y$=INKEY$ : IF Y$="" THEN GOTO 510
520 IF Y$="Y" OR Y$="y" THEN RUN "MENU"
530 IF Y$="N" OR Y$="n" THEN GOTO 550
540 BEEP : BEEP : GOTO 510
550 PRINT "ｺﾞｸﾛｳｻﾏ ﾃﾞｼﾀ"
560 END
```

一番目のプログラムは、ｙを入力しリターンキーを押したときにのみ「ＭＥＮＵ」が実行されます。２番目はリターンキーを押す必要もないうえ、無関係のキー入力があったときはブザー音を出して、もう一度正しい入力をうながします。お好きなほうをどうぞ。

プログラムの実行とリスト

```
10 ' file name ==> DMAIL2
20 ' 1984.6.20
30 ' random data correct
40 ' save "DMAIL2"
```

```
50 '
100 WIDTH 40,20 : CONSOLE 0,25,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
130 '
140 OPEN "R",#1,"1:DMAILDAT"
150 FIELD #1,20 AS NM$, 6 AS PN$,
             35 AS AD$,20 AS BA$
160 '
170 GOSUB 1000 : FLAG=1
180 DIM A(DM) : DM=0 : GOSUB 1000 : INDEX=DM
190 '
200 LOCATE 5,5 : FLAG=0
210 PRINT "ﾄﾞﾅﾀ ﾉ ｼｭｳｾｲ ﾃﾞｽｶ (END=9999)?"
220 LOCATE 7,6 : INPUT "ｺｰﾄﾞ ﾊﾞﾝｺﾞｳ";CD
230 IF CD=9999 THEN GOTO 300
240 LOCATE 10,8 : INPUT "OK ﾃﾞｽｶ ( y/n )";Y$
250 IF Y$="N" OR Y$="n" THEN CLS : GOTO 200
260 GOSUB 2000
270 CLS : GOTO 200
280 '
290 CLOSE #1
300 '
310 CLS
320 LOCATE 10,4 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 10,5 : INPUT "    ( y/n )";Y$
340 IF Y$="Y" OR Y$="y" THEN RUN "MENU03"
350 END
1000 '
1010 ' index read
1020 '
1030 OPEN "I",#2,"1:INDX1DAT"
1040 WHILE NOT EOF(2)
1050   DM=DM+1
1060   IF FLAG=1 THEN GOTO 1080
1070   INPUT #2,DUMMY : GOTO 1090
1080   INPUT #2,A(DM)
1090 WEND
1100 CLOSE #2
1110 RETURN
2000 '
2010 ' data correct sub
```

```
2020 '
2030 CLS
2040 FOR I=1 TO INDEX
2050   IF A(I)=CD THEN FLAG=1:SEC=I:I=INDEX
2060 NEXT I
2070 IF FLAG=0 THEN GOTO 2250
2080 GET #1,SEC
2090 LOCATE 2,3 : PRINT "ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ"
2100 LOCATE 4,4 : PRINT "ｼｭｳｾｲ ﾅｹﾚﾊﾞ RET ｷｰ"
2110 PRINT
2120 PRINT NM$ : INPUT "  ==> ";A1$
2130 IF A1$="" THEN A1$=NM$
2140 PRINT PN$ : INPUT "  ==> ";A2$
2150 IF A2$="" THEN A2$=PN$
2160 PRINT AD$ : INPUT "  ==> ";A3$
2170 IF A3$="" THEN A3$=AD$
2180 PRINT BA$ : INPUT "  ==> ";A4$
2190 IF A4$="" THEN A4$=BA$
2200 LSET NM$=A1$ : LSET PN$=A2$
2210 LSET AD$=A3$ : LSET BA$=A4$
2220 PUT #1,SEC
2230 RETURN
2240 '
2250 BEEP : BEEP : BEEP
2260 LOCATE 8,8 : PRINT "ｿﾉｶﾀ ﾊ ﾄｳﾛｸｼﾃ ｱﾘﾏｾﾝ"
2270 FOR I=1 TO 3000 : NEXT I
2280 RETURN
```

150ページで入力した「オブ　ノブナガ」を修正してみましょう。

3344というコード入力後の修正画面です。

```
  ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ
    ｼｭｳｾｲ ﾅｹﾚﾊﾞ RET ｷｰ

ｵﾀﾞ ﾉﾌﾞﾅｶﾞ
  ==> ?           ← 氏名は修正しないのでリターンキーを押す
231
  ==> ? 000       ← 新しい郵便番号
cccccccccc
  ==> ? xxxxxxxxxx ← 新住所
1984ﾊﾞﾝﾁ
  ==> ? 0-0-0     ← 番地の入力
```

4・4　宛て名の印刷

さて、いよいよ最後の印刷プログラムです。

ここで扱うプログラムは、ディスクに登録してあるデータすべての宛て名を印刷するのではなく、DMなどの発送の必要のある人だけをプリンタに出すことにします。

考え方は、対象となる人のコード番号をキーボードから入力し、それを一時、配列に退避させておいて、すべてのキー入力がすんだら、一括して印刷するシステムにします。

1件入力しては印刷する、というようにしてもかまわないのですが、大変時間の浪費になります。キー入力は人間がやらなければなりませんが、それさえすませたら、あとはプリンタにまかせましょう。

とはいっても、これは発送数が相当多数にわたる場合で、本節では2、3のデータをプリンタに打ち出してみるだけですから、プリント時間もつかの間ですが、一応発送数がふえることを想定して考えます。

しかし、データが多いか少ないかの相異だけで、基本的な考え方はまったく同じです。

プログラムヘッドはファイル名を変えるだけで、前節のそれと変わりません。

初期設定

```
10 ′ file name ==> DMAIL3
20 ′ 1984.6.20
30 ′ address print out
40 ′ save "DMAIL3"
50 ′
100 WIDTH 40,20 : CONSOLE 0,25,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
130 ′
140 OPEN "R",#1,"1:DMAILDAT"
150 FIELD #1,20 AS NM$, 6 AS PN$,
         35 AS AD$,20 AS BA$
```

ファイル名

次に**配列宣言とデータ読み込み**です。

```
160 ′
170 GOSUB 1000 : FLAG=1
180 DIM A(DM),B(DM):DM=0:GOSUB 1000:INDEX=DM
```

この部分のステートメントは、その持つ意味は前節と同じですが、ただ、配列Ｂを新しく加えています。

配列Ａはシーケンシャルファイルのインデックスをたくわえておくものでしたが、**配列Ｂ**は印刷する必要のあるデータのインデックスをしまっておきます。先ほど、まとめて印刷するといいましたが、つまり、**あとで配列Ｂにしまわれているコード番号を１つ１つ取り出し、それに対応するランダムファイルのセクタをＧＥＴし、プリンタに出力**します。

次は**コードの入力部分**です。　**コードの入力**

```
190 '
200 LOCATE 10,4 : PRINT "ｱﾃﾅ ﾉ ｲﾝｻﾂ ｦ ｼﾏｽ"
210 LOCATE 10,5 : PRINT "ｺｰﾄﾞ ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ"
220 LOCATE 10,6 : PRINT "        < END=9999 >"
230 LOCATE 10,8 : INPUT "  ==> ";CD
240 IF CD=9999 THEN GOTO 300
250 LOCATE 10,8 : PRINT "                    "
260 N=N+1
270 B(N)=CD
280 GOTO 230
```

すぐ前に入力したコードを消す（250行）

配列にコードをしまう（260〜270行）

230行でコードを入力するたびに、260行で配列Ｂの添字に使う変数Ｎを１ずつふやし、270行でコードをその配列に登録しています。

240行でエンドマークを判断すると、印刷の処理へ移ります。

```
290 '
300 GOSUB 2000
310 '
320 GOSUB 3000
330 '
340 CLS
350 LOCATE 10,4 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 10,5 : INPUT "     ( y/n )";Y$
370 IF Y$="Y" OR Y$="y" THEN RUN "MENU03"
380 END
```

インデックス検索サブルーチンへ（300行）

印刷サブルーチンへ（320行）

300行で2000行からのインデックス検索サブルーチンへ飛び、帰ってきて印刷サブルーチンへ飛びます。印刷が終わると、いつものとおり、ほかの作業をするかどうかのメッセージ表示となります。

次はサブルーチンを作りますが、**インデックスを読み込む1000から1100行のサブルーチンは156ページとまったく同じです**ので、そちらを見ながら入力してください。

インデックスの検索　2000行からのサブルーチンを見てみましょう。

```
2000 '
2010 '  find index sub
2020 '
2030 FOR I=1 TO N
2040   FOR J=1 TO INDEX
2050     IF B(I)=A(J) THEN B(I)=J : J=INDEX
2060   NEXT J
2070 NEXT I
2080 RETURN
```

ループ変数Ｉの最大値Ｎは、登録したデータの個数でしたね。260行で計算されています。

2040行から2060行へかけてのループは、先ほど入力したコードと、インデックスファイルに登録されているコードとを比較し、同じものが見つかれば、そのときのＪの値を配列Ｂにしまうわけです。つまりこの時点で、配列Ｂからそれまでに入っていたコード番号が追い出され、新しい数値（これが追い出されたコード番号に対応するデータのセクタ番号を示す）が入るわけです。

該当するコードが見つかれば、ループ変数Ｊの値を最大値にして、このループから抜け出します。余計な仕事はやめようというわけです。

この操作をＮ個のコードについて行えば、ＧＥＴするセクタがそれぞれわかったことになりますから、あとの印刷は簡単です。

<注>プリンタをお持ちでない方は3100行から3150行までのＬＰＲＩＮＴ文の「Ｌ」を削除し、3155行として時間つぶしのステートメントを入れてください。また、3040、3050行のメッセージは変えなければなりませんが、みなさんがくふうしてみてください。

印刷のサブルーチン

```
3000 '
3010 '  print out sub
3020 '
3030 CLS
3040 LOCATE 10,5 : PRINT "ｲﾝｻﾂ ｼﾏｽ"
3050 LOCATE 10,6 : PRINT "ﾌﾟﾘﾝﾀ ｦ ｾｯﾄｼﾃｸﾀﾞｻｲ"
3060 LOCATE 10,8 : PRINT "Hit any key if OK"
3070 Y$=INKEY$ : IF Y$="" THEN GOTO 3070
3080 FOR I=1 TO N
3090   GET #1,B(I)
3100   LPRINT TAB(5);PN$
3110   LPRINT TAB(5);AD$
3120   LPRINT TAB(15);BA$
3130   LPRINT
3140   LPRINT TAB(10);NM$;TAB(30);"ｻﾏ"
3150   FOR J=1 TO 5 : LPRINT : NEXT J
3160 NEXT I
```

3090行：バッファ上にデータをとりだす

3100〜3140行：印刷

3150行：前の宛て名との間に５行空間をとる

```
3170 CLOSE #1
3180 RETURN
```

3070行では、どのキーでも押せば印刷処理に移ります。

3150行は、すぐ前に印刷した宛て名との間に空白行を出力するステートメントです。

次に、プログラム総リストを示します。

```
10 ' file name ==> DMAIL3
20 ' 1984.6.20
30 ' address print out
40 ' save "DMAIL3"
50 '
100 WIDTH 40,20 : CONSOLE 0,25,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
130 '
140 OPEN "R",#1,"1:DMAILDAT"
150 FIELD #1,20 AS NM$, 6 AS PN$,
             35 AS AD$,20 AS BA$
160 '
170 GOSUB 1000 : FLAG=1
180 DIM A(DM),B(DM):DM=0:GOSUB 1000:INDEX=DM
190 '
200 LOCATE 10,4 : PRINT "ｱﾃﾅ ﾉ ｲﾝｻﾂ ｦ ｼﾏｽ"
210 LOCATE 10,5 : PRINT "ｺｰﾄﾞ ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ"
220 LOCATE 10,6 : PRINT "       < END=9999 >"
230 LOCATE 10,8 : INPUT "   ==> ";CD
240 IF CD=9999 THEN GOTO 300
250 LOCATE 10,8 : PRINT "             "
260 N=N+1
270 B(N)=CD
280 GOTO 230
290 '
300 GOSUB 2000
310 '
320 GOSUB 3000
330 '
340 CLS
350 LOCATE 10,4 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 10,5 : INPUT "    ( y/n )";Y$
370 IF Y$="Y" OR Y$="y" THEN RUN "MENU03"
380 END
```

```
1000 '
1010 '  index read
1020 '
1030 OPEN "I",#2,"1:INDX1DAT"
1040 WHILE NOT EOF(2)
1050   DM=DM+1
1060   IF FLAG=1 THEN GOTO 1080
1070   INPUT #2,DUMMY : GOTO 1090
1080   INPUT #2,A(DM)
1090 WEND
1100 CLOSE #2
1110 RETURN
2000 '
2010 '  find index sub
2020 '
2030 FOR I=1 TO N
2040   FOR J=1 TO INDEX
2050     IF B(I)=A(J) THEN B(I)=J : J=INDEX
2060   NEXT J
2070 NEXT I
2080 RETURN
3000 '
3010 '  print out sub
3020 '
3030 CLS
3040 LOCATE 10,5 : PRINT "ｲﾝｻﾂ ｼﾏｽ"
3050 LOCATE 10,6 : PRINT "ﾌﾟﾘﾝﾀ ｦ ｾｯﾄｼﾃｸﾀﾞｻｲ"
3060 LOCATE 10,8 : PRINT "Hit any key if OK"
3070 Y$=INKEY$ : IF Y$="" THEN GOTO 3070
3080 FOR I=1 TO N
3090   GET #1,B(I)
3100   LPRINT TAB(5);PN$
3110   LPRINT TAB(5);AD$
3120   LPRINT TAB(15);BA$
3130   LPRINT
3140   LPRINT TAB(10);NM$;TAB(30);"ｻﾏ"
3150   FOR J=1 TO 5 : LPRINT : NEXT J
3160 NEXT I
3170 CLOSE #1
3180 RETURN
```

次に示すのは、この章で扱ったデータをプリンタに出力したものです。あなたもためしてみてください。

また、いろいろ集計データや資料などを出力するルーチンを作り、宛て名書きをしたあとすぐに出力させるようにして、窓つき封筒を利用して発送すれば、かなり実用化できると思います。

```
030
bbbbbbbbbb
          456-78

     ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ          ｻﾏ


231
cccccccccc
          1984ﾊﾞﾝﾁ

     ｵﾀﾞ ﾉﾌﾞﾅｶﾞ          ｻﾏ


101
aaaaaaaaaa
          1-2-3

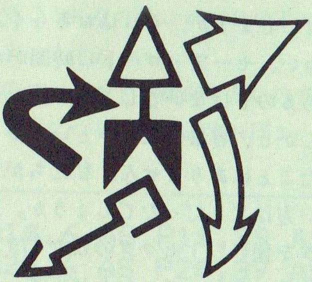
     ﾄｸｶﾞﾜ ｲｴﾔｽ          ｻﾏ


680
dddddddddd
          9-8-76

     ｱｹﾁ ﾐﾂﾋﾃﾞ           ｻﾏ
```

第5章

ランダムファイル[II]



5・1　ディスクを有効に使う

第４章の宛て名書きプログラム、いかがでしたか？　ディスクとのデータのやりとり、ご理解いただけたでしょうか。

ランダムデータファイル管理の基本は、おおよそ第４章に述べたとおりです。数値データを扱いませんでしたが、ＭＫＩ＄、ＭＫＳ＄、ＭＫＤ＄を使ってディスクに保存し、使いたいときにＣＶＩ、ＣＶＳ、ＣＶＤで数値に変換する、ということさえわかってしまえば、あとは加減乗除をしてプリンタに出力するとか、要するにディスクを使わない場合の操作と変わりありません。

ディスクを使ったらカセットテープはもう使う気にならない、とよく耳にしますが、それはおそらく、データを保存する容量がきわめて大きい、セーブ・ロードに時間がかからない、といったあたりに理由があるのではないでしょうか。

しかし、容量が大きいといっても、できるだけ有効に利用するにこしたことはありません。私たちが学んだ、前章でのデータファイルの使い方はどうだったでしょうか。氏名、郵便番号、住所、番地で81セクタを使い、１セクタの残り175バイトは未使用でした。ずいぶんムダな使い方をしていたわけです。

この章では、ディスクの有効利用をテーマに話を進めますが、できたら話につながりがあったほうがよいと思いますので、前章のプログラムを例にとって、一部を変更し、一部ステートメントを追加し、ご説明したいと思います。

ムダのないデータ設計

〈注〉前章では住所に35バイトを割り当てていましたが、ここでは４バイト分プラスして39バイトとします。

前章では１セクタ内に１名分のデータを書き込んでいました。

今度は下の図のように３名分のデータを保存することにしましょう。１名で85バイト使いますから、３名で255バイト、つまり１バイトだけ未使用ということになります。ほぼ100パーセント活用です。

氏　名(20バイト)	郵便番号(6バイト)	住　所(39バイト)	番 地(20バイト)
氏　名(20バイト)	郵便番号(6バイト)	住　所(39バイト)	番 地(20バイト)
氏　名(20バイト)	郵便番号(6バイト)	住　所(39バイト)	番 地(20バイト)

← 1バイト未使用

さて、1セクタ内のデータフォーマットが決まりましたから、まず問題になるのはフィールドの割り当てです。

前章で、

```
FIELD #1,20 AS NM$,6 AS PN$,39 AS AD$,20 AS BA$
```

のように割り当てていましたので、同じ考え方をして次のように定義すればよいのでしょうか？

```
FIELD #1,20 AS N1$,6 AS P1$,39 AS A1$,20 AS B1$
FIELD #1,20 AS N2$,6 AS P2$,39 AS A2$,20 AS B2$
FIELD #1,20 AS N3$,6 AS P3$,39 AS A3$,20 AS B3$
```

一見だまされてしまいそうな感じですが、この場合、3人分の各項目が同じＦＩＥＬＤを共有していることになり、これでＰＵＴ、ＧＥＴをすると、データが混同してしまいます。

ですから、バッファ上に3人分のフィールドを確保したいのでしたら、すべてコンマで区切って次のようにつなげて定義すればよいのです。

Ⓐ

```
FIELD #1,20 AS N1$,6 AS P1$,39 AS A1$,20 AS B1$
        ,20 AS N2$,6 AS P2$,39 AS A2$,20 AS B2$
        ,20 AS N3$,6 AS P3$,39 AS A3$,20 AS B3$
```

そしてこれを少し変形して次のようにしても、それぞれの氏名、郵便番号、住所、番地に応じたフィールド幅を確保することができます。

Ⓑ

```
❶FIELD #1,  0 AS D$,20 AS N1$,6 AS P1$,39 AS A1$,20 AS B1$
❷FIELD #1, 85 AS D$,20 AS N2$,6 AS P2$,39 AS A2$,20 AS B2$
❸FIELD #1,170 AS D$,20 AS N3$,6 AS P3$,39 AS A3$,20 AS B3$
```

これはどういうことかと申しますと、まず❶のＦＩＥＬＤ文で、先頭のＤ＄は0ですから、先頭にＮ１＄が入り順次Ｂ１＄まで設定されます。次に❷であらためてＦＩＥＬＤ文が定義され、まず85バイトのＤ＄が入り、そのうしろに続いてＮ２＄～Ｂ２＄が設定されます。

ダミー変数を使う

ここに登場する**Ｄ＄はダミー**で、実際には何も利用されないものです。そしてこの85バイトという数は、❶のＦＩＥＬＤ文で定義されている変数のフィールド幅を合計したものです。

つまり、❶でＮ１＄～Ｂ１＄のフィールド域が確保され、❷ではそのフィールド幅をもったダミーを踏み台にしてＮ２＄～Ｂ２＄のフィールド域が決められています。ということはバッファ上ではＮ１＄からＢ２＄までのフィールド域が作られたということになります。

同じように、❸のＦＩＥＬＤ文によって、Ｎ３＄からＢ３＄までが確保されたことになります。

❶～❸の過程で、ダミーのＤ＄は０、85、170バイトと変化し、最終的には170バイトが残ります。

しかしこのＤ＄、まったくのダミーですから、プログラムでＧＥＴ、ＰＵＴさえしなければよいのです。

さて、これでだいぶＦＩＥＬＤ文の下準備ができてきました。

要は、ダミー変数を使えば、そのうしろに希望のフィールド域を確保できるということを理解していただけばよいと思います。

ここでもう一度前に戻って、Ⓐ のＦＩＥＬＤ文を次のように配列を使って書き換えてみましょう。

```
FIELD #1,20 AS N$(1),6 AS P$(1),39 AS A$(1),20 AS B$(1)
        ,20 AS N$(2),6 AS P$(2),39 AS A$(2),20 AS B$(2)
        ,20 AS N$(3),6 AS P$(3),39 AS A$(3),20 AS B$(3)
```

そしてⒷも書き換えてみます。

```
FIELD #1,  0 AS D$,20 AS N$(1),6 AS P$(1),39 AS A$(1),20 AS B$(1)
FIELD #1, 85 AS D$,20 AS N$(2),6 AS P$(2),39 AS A$(2),20 AS B$(2)
FIELD #1,170 AS D$,20 AS N$(3),6 AS P$(3),39 AS A$(3),20 AS B$(3)
```

もうおわかりと思います。

いくらなんでも同じ形のＦＩＥＬＤ文を３行も長々と書くのはちょっとナンセンスですね。ＦＯＲ～ＮＥＸＴ文でもう少しすっきりさせたいものです。そうです、次のように表せますね。

```
200 FOR I=1 TO 3
210    FIELD #1,85*(I-1) AS D$,20 AS N$(I)
        6 AS P$(I),39 AS A$(I),20 AS B$(I)
220 NEXT I
```

ずいぶんさっぱりした形に表現できました。

このように、バッファをいくつかのレコードで使用する場合には、文字変数をたくさんつらねたりしないで、ＦＯＲ～ＮＥＸＴ文によって処理できるのです。それに、ダミーの考え方、これも大切ですからぜひ体得してください。

さて、前章では１セクタに１レコードを書き込んでいましたが、この例では１セクタに３レコードを書き込めます。すると、ＰＵＴ、ＧＥＴはどうしたらよいのでしょうか？

GET、PUTするセクタ番号、そのセクタ内で何番目のレコードかがわかればよいわけですから、インデックスファイルのレコード番号を３で割った商、余りを利用することによって求められます。この辺は具体例で学ぶことにしましょう。

これから、前章で作ったプログラムを一部変更・追加して、１セクタに３名分書き込めるようにしようというわけですが、**一番大きな変更点はＦＩＥＬＤ文**です。

そして、**いちばん大切な追加点はセクタ番号、セクタ内でのレコード番号を計算するステートメント**です。

すでにでき上がっているプログラムに手を加えてということですから、条件文など、多少まわりくどい点もやむをえないと思います。

しかし、個人で、自分の仕事に合ったプログラムを作ろうとするとき、やはり最初に小さなプログラムを作って、次に、あんなこともしたい、こんなこともしたいと欲がでてきて、変更・追加をくりかえして、プログラムをふくらませていくのです。

それでは、プログラムの手直しにかかりますが、順を追っての説明はやめ、修正後の全プログラムリストをかかげ、**追加点は緑色の文字で、修正個所は緑色のアミで**示すことにしました。

データ書き込みプログラム

```
10 '  file name ==> dmail1
20 '  1984.6.20
30 '  random data write
40 '  save "dmail1"
50 '
100 WIDTH 40,20 : CONSOLE 0,25,0,0
110 CLEAR 5000 : DEFINT A-Z
115 DIM NM$(3),PN$(3),AD$(3),BA$(3)
120 CLS
130 '
140 '  open file
150 OPEN "R",#1,"1:dmaildat"
155 FOR I=1 TO 3
160 FIELD #1,85*(I-1) AS DUMMY$,20 AS NM$(I),
         6 AS PN$(I),39 AS AD$(I),20 AS BA$(I)
165 NEXT I
170 IF LOF(1)<>0 THEN GOSUB 1000 : GOTO 200
```

（115行）配列宣言をしてメモリを有効に使う

```
180 OPEN "O",#2,"1:indx1dat"
190 P=1 : GOTO 220
200 OPEN "A",#2,"1:indx1dat"
210 '
220 ' data input
230 LOCATE 10,5 : PRINT "ﾃﾞｰﾀ ﾉ ﾆｭｳﾘｮｸ ｦ ｼﾏｽ"
240 FOR I=1 TO 3000 : NEXT I
250 CLS
260 PRINT "ｺｰﾄﾞ No. ｦ ﾄﾞｳｿﾞ (END=9999)"
270 INPUT "    ==> ";CD
280 IF CD=9999 THEN GOTO 490
290 PRINT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ"
300 INPUT "    ==> ";A1$
310 PRINT "ﾕｳﾋﾞﾝ ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ"
320 INPUT "    ==> ";A2$
330 PRINT "ｼﾞｭｳｼｮ ｦ ﾄﾞｳｿﾞ"
340 INPUT "    ==> ";A3$
350 PRINT "ﾊﾞﾝﾁ ｦ ﾄﾞｳｿﾞ"
360 INPUT "    ==> ";A4$
370 PRINT
380 INPUT "      OK ﾃﾞｽｶ ( y/n )";Y$
390 IF Y$="N" OR Y$="n" THEN GOTO 250
400 '
410 ' data write
412 SEC=(P-1) ¥ 3+1          ← セクタ番号の計算
414 REC=(P-1) MOD 3+1        ← セクタ内でのレコード番号を計算
416 IF REC=1 THEN GOTO 420
418 GET #1,SEC
420 PRINT #2,CD
430 LSET NM$(REC)=A1$ : LSET PN$(REC)=A2$
440 LSET AD$(REC)=A3$ : LSET BA$(REC)=A4$
450 PUT #1,SEC
460 P=P+1                    ← レコードの通し番号を加算
470 GOTO 250
480 '
490 ' file close
500 CLOSE #1,#2
510 '
520 CLS
530 LOCATE 10,4 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
540 LOCATE 10,5 : INPUT "   ( y/n )";Y$
550 IF Y$="Y" OR Y$="y" THEN RUN "MENU04"
```

```
560 END
1000 '
1010 ' data count sub
1020 OPEN "I",#1,"1:indx1dat"
1030 WHILE NOT EOF(2)
1040   INPUT #2,DUMMY
1050   P=P+1 ←―――――― レコード数の加算
1060 WEND
1070 P=P+1
1080 CLOSE #2
1090 RETURN
```

前章のプログラム「DMAIL1」をロードして、色アミの部分を修正し、緑色のステートメントを追加入力してください。

●修正・追加が終ったら新ファイル名でセーブするのを忘れないようにしてください。

115行のDIM文は、FIELD文の定義に使う変数のための配列ですが、これはなくてもかまいません。配列を宣言しなければ、自動的に10個分は確保されますが、このように、使用する大きさだけ宣言しておけば、メモリの節約になると考えたからです。

155行から165行のフィールド設定、もうおわかりですね。

412行、414行で、セクタ番号（SEC)、セクタ内でのレコード番号（REC）を決定しています。

セクタ番号（SEC）は、レコードの通し番号（P）から1を引いた数を3で割り、その商に1を加算して計算します。

〈注〉412行の円マーク「¥」は整数除算の商を求めるものです。

セクタ内でのレコード番号（1番目のレコードか、2番目のレコードか、3番目のレコードか、ということ）RECは、インデックスファイルでのレコードの通し番号（P）から1を引いた値を3で割り、その余り（0か1か2のいずれか）に1を加算して決めます。

ここでいうレコードの通し番号は、初回の書き込みであれば190行で「1」に設定され、追加モードの場合には1000行からのサブルーチンで決定されていますね。つまり、この値を使ってセクタ内でのレコード番号を算出しようというわけです。算出には「MOD」を使っていますが、これは整数除算の余りを求めるものです。

次に、416行では「もしセクタ内でのレコード番号が1のときには420行へジャンプ」というステートメントにしていますが、これは、418行で新しいセクタをGETしようとすると「Input Past End」のエラーになってしまうからです。

418行のGET文はなぜ入れておくかといいますと、450行で1セクタ分まるまるPUTするわけですから、ここでGETしてすでにファイルに書き込まれている同じセクタ内のデータをバッファ上に取り出

しませんと、450行のＰＵＴ文で消えてしまいます。

たとえば、あるセクタの第１、第２レコードはすでに書き込まれていたとします。そして今回３番目のレコードを入力しました。このとき、このセクタをＧＥＴして第１、第２レコードもバッファ上に取り出しませんと、バッファ上の第１、第２レコードに割り当てられた変数には何も入っていませんから、この状態でＰＵＴすると、第３レコードだけが書き込まれ、第１、第２レコードはクリアされてしまいます。このミスはよくありますので注意してください。

460行ではレコード番号を１進めています。

次にデータ修正のプログラムに移りますが、変更・追加個所は、いま説明したようなところです。プログラムを追ってみてください。

データ修正のプログラム

```
10 ' file name ==> dmail2
20 ' 1984.6.20
30 ' random data correct
40 ' save "dmail2"
50 '
100 WIDTH 40,20 : CONSOLE 0,25,0,0
110 CLEAR 5000 : DEFINT A-Z
115 DIM NM$(3),PN$(3),AD$(3),BA$(3)
120 CLS
130 '
140 OPEN "R",#1,"1:dmaildat"
145 FOR I=1 TO 3
150 FIELD #1,85*(I-1) AS DUMMY$,20 AS NM$(I),
      6 AS PN$(I),39 AS AD$(I),20 AS BA$(I)
155 NEXT I
160 '
170 GOSUB 1000 : FLAG=1
180 DIM A(DM) : DM=0 : GOSUB 1000 : INDEX=DM
190 '
200 LOCATE 5,5 : FLAG=0
210 PRINT "ドナタ ノ シュウセイ デスカ (END=9999)?"
220 LOCATE 7,6 : INPUT "コード バンゴウ";CD
230 IF CD=9999 THEN GOTO 300
240 LOCATE 10,8 : INPUT "OK デスカ ( y/n )";Y$
250 IF Y$="N" OR Y$="n" THEN CLS : GOTO 200
260 GOSUB 2000
270 CLS : GOTO 200
```

```
280 '
290 CLOSE #1
300 '
310 CLS
320 LOCATE 10,4 : PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 10,5 : INPUT "   ( y/n )";Y$
340 IF Y$="Y" OR Y$="y" THEN RUN "MENU04"
350 END
1000 '
1010 '  index read
1020 '
1030 OPEN "I",#2,"1:indx1dat"
1040 WHILE NOT EOF(2)
1050   DM=DM+1
1060   IF FLAG=1 THEN GOTO 1080
1070   INPUT #2,DUMMY : GOTO 1090
1080   INPUT #2,A(DM)
1090 WEND
1100 CLOSE #2
1110 RETURN
2000 '
2010 '  data correct sub
2020 '
2030 CLS
2040 FOR I=1 TO INDEX
2050   IF A(I)=CD THEN FLAG=1 : P=I : I=INDEX
2060 NEXT I
2070 IF FLAG=0 THEN GOTO 2250
2072 SEC=(P-1) ¥ 3+1
2074 REC=(P-1) MOD 3+1
2080 GET #1,SEC
2090 LOCATE 2,3 : PRINT "ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ"
2100 LOCATE 4,4 : PRINT "ｼｭｳｾｲ ﾅｹﾚﾊﾞ RET ｷｰ"
2110 PRINT
2120 PRINT NM$(REC) : INPUT "   ==> ";A1$
2130 IF A1$="" THEN A1$=NM$(REC)
2140 PRINT PN$(REC) : INPUT "   ==> ";A2$
2150 IF A2$="" THEN A2$=PN$(REC)
2160 PRINT AD$(REC) : INPUT "   ==> ";A3$
2170 IF A3$="" THEN A3$=AD$(REC)
2180 PRINT BA$(REC) : INPUT "   ==> ";A4$
2190 IF A4$="" THEN A4$=BA$(REC)
```

```
2200 LSET NM$(REC)=A1$ : LSET PN$(REC)=A2$
2210 LSET AD$(REC)=A3$ : LSET BA$(REC)=A4$
2220 PUT #1,SEC
2230 RETURN
2240 '
2250 BEEP : BEEP : BEEP
2260 LOCATE 8,8 : PRINT "ｿﾉｶﾀ ﾊ ﾄｳﾛｸｼﾃ ｱﾘﾏｾﾝ"
2270 FOR I=1 TO 3000 : NEXT I
2280 RETURN
```

印刷プログラム

```
10 '  file name ==> dmail3
20 '  1984.6.20
30 '  address print out
40 '  save "dmail3"
50 '
100 WIDTH 40,20 : CONSOLE 0,25,0,0
110 CLEAR 5000 : DEFINT A-Z
115 DIM NM$(3),PN$(3),AD$(3),BA$(3)
120 CLS
130 '
140 OPEN "R",#1,"1:dmaildat"
145 FOR I=1 TO 3
150 FIELD #1,85*(I-1) AS DUMMY$,20 AS NM$(I),
       6 AS PN$(I),39 AS AD$(I),20 AS BA$(I)
155 NEXT I
160 '
170 GOSUB 1000 : FLAG=1
180 DIM A(DM),B(DM):DM=0:GOSUB 1000:INDEX=DM
190 '
200 LOCATE 10,4 : PRINT "ｱﾃﾅ ﾉ ｲﾝｻﾂ ｦ ｼﾏｽ"
210 LOCATE 10,5 : PRINT "ｺｰﾄﾞ ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ"
220 LOCATE 10,6 : PRINT "      < END=9999 >"
230 LOCATE 10,8 : INPUT "  ==> ";CD
240 IF CD=9999 THEN DIM REC(N) : GOTO 300
250 LOCATE 10,8 : PRINT "          "
260 N=N+1
270 B(N)=CD
280 GOTO 230
290 '
300 GOSUB 2000
```

セクタ内でのレコード番号を格納するための配列

```
301 FOR I=1 TO N
302   REC(I)=(B(I)-1) )MOD 3+1 ←セクタ内でのレコード番号
303   B(I)=(B(I)-1) )¥ 3+1 ←そのデータが属するセクタ番号
304 NEXT I
310 '
320 GOSUB 3000
330 '
340 CLS
350 LOCATE 10,4 : PRINT "ホカ ノ シコ゛ト ヲ シマスカ"
360 LOCATE 10,5 : INPUT "     ( y/n )";Y$
370 IF Y$="Y" OR Y$="y" THEN RUN "MENU04"
380 END
1000 '
1010 '  index read
1020 '
1030 OPEN "I",#2,"1:indx1dat"
1040 WHILE NOT EOF(2)
1050   DM=DM+1
1060   IF FLAG=1 THEN GOTO 1080
1070   INPUT #2,DUMMY : GOTO 1090
1080   INPUT #2,A(DM)
1090 WEND
1100 CLOSE #2
1110 RETURN
2000 '
2010 '  find index sub
2020 '
2030 FOR I=1 TO N
2040   FOR J=1 TO INDEX
2050     IF B(I)=A(J) THEN B(I)=J : J=INDEX
2060   NEXT J
2070 NEXT I
2080 RETURN
3000 '
3010 '  print out sub
3020 '
3030 CLS
3040 LOCATE 10,5 : PRINT "インサツ シマス"
3050 LOCATE 10,6 : PRINT "フ゜リンタ ヲ セットシテクタ゛サイ"
3060 LOCATE 10,8 : PRINT "Hit any key if OK"
3070 Y$=INKEY$ : IF Y$="" THEN GOTO 3070
```

```
3080 FOR I=1 TO N
3090   GET #1,B(I)
3095   P=REC(I)
3100   LPRINT TAB(5);PN$(P)
3110   LPRINT TAB(5);AD$(P)
3120   LPRINT TAB(15);BA$(P)
3130   LPRINT
3140   LPRINT TAB(10);NM$(P);TAB(30);"ｻﾏ"
3150   FOR J=1 TO 5 : LPRINT : NEXT J
3160 NEXT I
3170 CLOSE #1
3180 RETURN
```

メニュープログラム

```
10 ' file name ==> MENU04
20 ' 1984.6.20
30 ' menu program for address printing
40 ' save "MENU04"
50 '
100 WIDTH 40,25 : CONSOLE 0,25,0,0
110 DEFINT A-Z
120 CLS
130 '
140 READ MENU
150 DIM M$(MENU)
160 FOR I=1 TO MENU
170   READ M$(I)
180 NEXT I
190 '
200 LOCATE 7,4 : PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,5:PRINT "|ｱﾃﾅ ｲﾝｻﾂ ﾌﾟﾛｸﾞﾗﾑ No.2|"
230 LOCATE 7,6 : PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270   LOCATE 8,7+I : PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13 : INPUT "input job number ";NO
```

```
310 LOCATE 8,15 : PRINT M$(NO)
320 LOCATE 18,16 : INPUT "OK (y/n)";Y$
330 IF Y$="N" OR Y$="n" THEN CLS : GOTO 200
340 ON NO GOTO 360,370,380,390
350 '
360 RUN "dmail1"
370 RUN "dmail2"
380 RUN "dmail3"
390 CLS
400 LOCATE 15,7 : PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
410 END
1000 '
1010 '  data area
1020 '
1030 DATA 4
1040 DATA ﾃﾞｰﾀ ﾉ ﾆｭｳﾘｮｸ･･･････1
1050 DATA ﾃﾞｰﾀ ﾉ ｼｭｳｾｲ････････2
1060 DATA ｱﾃﾅ ﾉ ｲﾝｻﾂ･･････････3
1070 DATA ｵﾜﾘ･････････････････4
```

5・2 マスターメニュープログラム

これで『ディスクファイル入門(1)』は終りです。

FM－77の内蔵ディスクを活用しての実用化という点に重点をおきましたので、プログラムのテーマはそれほどおもしろくなかったと思います。しかし、本書の目的は、ファイルの扱いを理解し、データをどう保存するか、そして各種の処理プログラムをどう連結するかという点にありましたので、その意味では、まだ入門段階ではありますが、一応の使命ははたせたのではないかと思います。

これをヒントに、あなたの仕事に合ったプログラムの構築をされることを期待します。

さあ、それではメニュープログラムのためのメニュープログラム、つまりマスター版を作っておしまいです。これをRUNさせて、いままでのプログラムが正常に走るかどうかテストしてみてください。

```
10 ' file name ==> MASTER
20 ' 1984.7.1
30 ' master menu program
40 ' save "MASTER"
50 '
100 WIDTH 40,25 : CONSOLE 0,25,0,0
110 CLEAR 5000 : DEFINT A-Z
120 CLS
130 '
140 READ MENU
150 DIM M$(MENU)
160 FOR I=1 TO MENU
170   READ M$(I)
180 NEXT I
190 '
200 LOCATE 7,2 : PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,3:PRINT "│  ﾏｽﾀｰﾒﾆｭｰ ﾌﾟﾛｸﾞﾗﾑ    │"
230 LOCATE 7,4 : PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
```

```
260 FOR I=1 TO MENU
270   LOCATE 8,5+I : PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13 : INPUT "ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ";NO
310 LOCATE 8,15 : PRINT M$(NO)
320 LOCATE 18,16 : INPUT "OK <y/n>";Y$
330 IF Y$="N" OR Y$="n" THEN CLS : GOTO 200
340 ON NO GOTO 360,370,380,390,400
350 '
360 RUN "MENU01"
370 RUN "MENU02"
380 RUN "MENU03"
390 RUN "MENU04"
400 CLS
410 LOCATE 15,7 : PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
420 END
1000 '
1010 ' data area
1020 '
1030 DATA 5
1040 DATA ｴｲﾀﾝｺﾞ ﾉ ﾚﾝｼｭｳ······1
1050 DATA ﾃﾞﾝﾜﾁｮｳ·············2
1060 DATA ｱﾃﾅ ﾉ ｲﾝｻﾂ··········3
1070 DATA ｱﾃﾅ ﾉ ｲﾝｻﾂ No.2·····4
1080 DATA ｵﾜﾘ··················5
```

5·3 ファイルをコピーする

私たちは、第１章でオリジナルのシステムディスクに対してＦＩＬＥＳコマンドを実行し、ディスクに保存されているプログラムなどを画面に表示させましたが、この中には、あまり使わないものがたくさんあります。

そこで、日常、比較的使いそうなプログラム（ファイル）だけを、ディスクコードだけをコピーしたディスク(24ページの❸のディスク)に移しておくことにしましょう。

なお、ここでのディスク名は、24ページに示した❷、❸などの黒マル数字で呼びますのでご諒承ください。

あるディスクにおさめられているプログラム（ファイル）を他のディスクに移すことを**「ファイルをコピーする」**または**「ファイルを転送する」**といいます。

この役目を果たしてくれるのが、コピー版システムディスクにおさめられている**「ＦＣＯＰＹ」**というプログラムです。

さあ、それではさっそく作業にとりかかりましょう。

まず、**❷のディスクをドライブ０に、❸のディスクをドライブ１にセット**してください。

次に「ＦＣＯＰＹ」をロードして実行します。

```
RUN "FCOPY" ↵
```

すると、次のメッセージが表示されます。

```
*** FILE COPY ***

SOURCE FILE DESCRIPTOR=■
```

これは「(転送する)**元になるファイルディスクリプタを入力してください**」と問いかけているのです。ファイルディスクリプタについてわからない人は17ページをごらんください。

最初に**「ＤＦＭＣＤ」をコピーする**ことにしましょう。

次のようにキーインしてください。

```
"0:DFMCD" ↵
```

ドライブ０のファイルに対しては「０：」は省略してもかまいませ

んが、ダブルクォーテーションは必ず入れてください。

すると、次のメッセージが表示されます。

```
DESTINATION FILE DESCRIPTOR=■
```

今度は「転送先でのファイルディスクリプタ」の入力を求めています。

私たちはドライブ1のディスクに、またファイル名は変更しないことにしますので、次のように入力します。

```
"1:DFMCD" ⏎
```

すると、次のようにOKかどうか確認してきます。

```
READY ? ■
```

だいじょうぶでしたら「Y」をキーインして ⏎ です。

間もなくすると「ドライブ0」から「ドライブ1」のディスクへ「DFMCD」が転送されて、次のような表示になります。

```
FILE COPY FINISHED !!
COPY ANOTHER FILE(Y/N) ? ■
```

ファイルのコピーをこれで終りにするなら「N」をキーインすればよいのですが、私たちはもう少し作業が残っています。Y ⏎ とキーインしましょう。

すると再びファイルディスクリプタの入力になりますから、同じ要領で次に示すファイルをすべてコピーしてください。

```
"0:MCOPY"
"0:SYSDSK"
"0:VOLCOPY"
"0:AUTOUTY"
"0:PFDEF"
"0:FCOPY"
"0:FCOPYM"
"0:FCOPYEB"
```

そして、最後のファイルのコピーが終ったら「N」をキーインしてこの作業を終りにしましょう。

さて、本当にコピーできたかどうか確認してみましょう。

```
FILES "1:"
```

画面は次のようになりました。

```
FILES "1:"

DFMCD      2 B S 1
MCOPY      2 B S 1
SYSDSK     0 B S 1
VOLCOPY    0 B S 1
AUTOUTY    0 B S 1
PFDEF      0 B S 2
FCOPY      0 B S 1
FCOPYM     0 B S 3
FCOPYEB    2 B S 1

140 Clusters Free

Ready
█
```

うまくいったようですね。

それでは、ドライブ0側のディスク❷を取り出し、ドライブ1側の❸のディスクをドライブ0にセットしましょう。

❷のディスクは「オリジナルのシステムディスク」のコピー版として大切に保管しておきましょう。

なお、一度に複数個のファイルを転送コピーするときは、ユーティリティプログラム

FCOPYM

を使うと便利です。あとでためしてみてください。

FM-77ディスクファイル入門(1)

著　者	相場多聞	©Tamon Aiba　1984
発行者	田村正隆	
発行所	株式会社　ナツメ社 東京都千代田区神田神保町１－52(〒101) 電話　03(291)1257(代表) 振替　東京３－58661	
印　刷	ラン印刷社	
製　本	三修紙工	

ISBN4-8163-0403-7

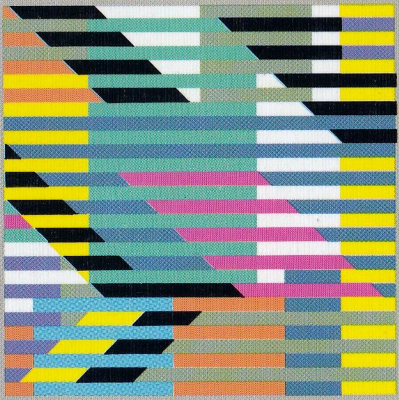
FM-77

ディスクファイル入門[1]

ナツメ社

定価=1400円

[目次より]



ISBN4-8163-0403-7 C2054 ¥1400E